Revenue Improvement Through Demand-Dependent Pricing of Network Services

A Thesis Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment of the Requirements for the Degree
Master of Science (Systems Engineering)

David M. Sanders

MAY 2000

ABSTRACT

This thesis addresses the issue of efficient resource allocation in broadband networks through the pricing of guaranteed services. Resource allocation in networks is typically achieved through technical methods such as scheduling, routing, and admission control, however these techniques are not predictive in terms of the expectation of rewards based upon variable demands. This work shows that revenue improvement can occur in this network environment when a dynamic pricing policy is applied as opposed to optimal static pricing. The network model used is that of a broadband channel with multiple classes of service, each of which has a Poisson arrival process with exponential service times, where the arrival rate is a function of the current price and a partially observed Markov chain. The optimization model is a continuous time, average-reward, partially observed Markov decision process, with the state defined as the profile of current active sessions (of various types) and the probability distributions over the demand states for each service class. We have developed a heuristic that seeks improvement of the objective based upon the probability that the service class is in one of several possible demand states. Our heuristic involves a state estimation procedure to determine (1) when the system has transitioned to a new demand state and (2) when and what control (price) should be applied to improve revenue.

This work contributes to the large body of recent research into network pricing which primarily examines the concept from a static point of view. Most analysis in this area of research assume a constant load upon the system and does not explicitly accommodate the stochastic nature of arrivals and departures, as does our heuristic. In the more common network pricing formulation, demand is assumed to be a constant and known function of price, and prices are adjusted to find an optimal mix of the user classes. In contrast, our formulation takes into account the stochastic nature of demand and takes advantage of the gains in revenue which can be made from fluctuations in demand.

Our heuristic has been "tested" against optimal static prices in several scenarios of varying demand. Our computational study involves a broadband network providing two distinct service classes and has shown the potential for revenue increase between 10% and 78%. It is surprising that significant gains such as these are possible given that in each scenario tested, any particular demand state profile (interpreted as a perfect information problem) exhibits very little gain in dynamic pricing. Additionally, the state estimation procedure we have developed is capable of correct classification rates as high as 96%, with the lowest observed rate being 62%, at least for the scenarios we have tested.

ACKNOWLEDGEMENTS

I would like to express sincere gratitude to my Advisor, Stephen Patek, without whom this thesis would not have been completed. His advice, encouragement, and complete understanding of the subject were indispensable. He was always available to assist me and that has made a substantial difference in the quality of this work.

I would also like to thank my dearest wife, Monique, who has always supported me whole-heartedly in my endeavors. Without her not only this thesis but most of the accomplishments in my life would not have been completed.

Table of Contents

Table of Figures

Table of Tables

List Of Symbols

λ_i:	Arrival rate of class of service i
$\lambda_{0,I}$:	Maximum arrival rate for class i
a_i b_i a^c_i b^c_i:	demand function parameters
μ_i:	service rate for class i
N(t)	$= \{n_1(t),\ldots,n_M(t)\}$ System state representation, number of users, class i , in system at time t
M	Number of service classes
u_i	price of service, class i
R	Link bandwidth [Mbps]
R_i	effective bandwidth requirement, class i [Mbps]
$\Pi(N(t))$	Pricing policy, dependent on system state
ds_i	demand state of class i
ν	maximum transition rate of Markov Chain
$g_{(k,l)}(\tau)$	probability of transition from ds=k to ds=i during time period τ
$f_{(k,u)}(\tau)$	probability of arrival occurring during time τ, given ds=k and price = u (if an arrival has occurred); probability of no arrival occurring during time τ, given ds=k and price = u (if a departure has occurred)
$\alpha_{i,j}$	transition rate of demand state, class i, from state j to state j+1
$\beta_{i,j}$	transition rate of demand state, class i, from state j to state j-1
$\overline{\alpha}_i$	maximum (increasing) demand transition rate, class i
$\overline{\beta}_i$	maximum (decreasing) demand transition rate, class i

Chapter 1. Introduction

1.1 Motivation

Computer networks were originally developed for the Department of Defense (DoD) in the mid 1960s to provide a secure command and control system in the event of nuclear war [T96]. Since that time the use of computer networks has grown at an amazing pace, with the current number of users more than doubling each year [GSW95a]. Networks are now a significant part of the daily lives of many people and the expanded role of computers and their networks in applications such as email, file sharing, World Wide Web, and video teleconferencing have irrevocably changed both our business and personal lives. As these networks become more involved in the business world they must adapt to the requirements of normal business practices – such as the necessity to be cost effective and efficient. In order to become cost effective companies which own these networks must have a mechanism for controlling the use of the resource to ensure it is not 'wasted', much in the same manner that any other resource is managed. In this way applications that are more beneficial to a business's bottom line will be admitted onto a network before applications that are less profitable.

This thesis deals with the issue of efficient resource allocation in broadband networks. The act of controlling the use of the network resource, bandwidth, is through bandwidth management. Currently this is achieved through technical mechanisms such as scheduling, routing, and admission control, however these measures are limited in the face of growing demand. The increase in demand for the transport of data on the network causes congestion, which slows the network and at times can cause the network to fail

[T96]. When the network becomes congested both the throughput rate and transit delay are adversely effected. This causes the quality and usefulness of inelastic applications (which are delay sensitive, such as real time audio and video) to become significantly degraded. In order for these applications to be effective we must be able to manage the available bandwidth to accommodate the desired applications Quality of Service (QoS).

In particular we are interested in developing a methodology that will allow the use of inelastic applications such as Vide-On-Demand (VOD) or Video Tele-conferencing (VTC). While these services are currently available in various formats, they are severely limited by the congestion experienced on data networks. While a VTC application can work through almost any Internet Service Provider the quality of the service can vary tremendously as conditions upon the underlying network change dynamically under a bursty traffic load. This degraded performance may be satisfactory for a grandparent to speak and see grandchildren through a VTC on a Saturday morning, however it can cause difficulties in business and academic conferences on busier times of the week. We believe that it is worthwhile to enhance this service so that routine, high quality VTC is available for the people and businesses willing to pay the cost of reserving the necessary resources.

VOD also currently works in some instances – such as inside hotels or on very small, privately owned distribution networks. The cost of this service runs around $10.00 a movie, and generally the service cannot be adjusted after it begins playing (in the version of VOD discussed by [T96], for example, you can pause a movie or rewind, or do anything you would be able to do if you had rented a video cassette and put it in your VCR). The capability does not currently exist to expand this enhanced service into a medium size community, even though the technology exists to do so [T96]. The reason that it cannot be used in a community setting is because of the cost of improving the physical lines into the homes to increase bandwidth and because of the network congestion that would result from its use. The cost from the vendor standpoint could be kept to somewhere around $4.00 a movie [T96] – so if we could control the congestion which such a service would generate it could make it quite a bit easier to watch a recently released movie for around the same price it costs now at the local rental store.

1.2 Problem definition

This thesis develops a methodology to allocate bandwidth in order to allow multiple types of traffic, with associated QoS parameters, to operate efficiently on a single network. The technique we will use is that of a pricing mechanism for services, which will affect demand, and through the use of admission control. Our research seeks to expand work done recently by [PT99] in which a network resource allocation problem was cast as a dynamic programming model in order to maximize the revenue of the network owner through a pricing mechanism that influenced demand. One of the strongest results of their work showed that in the case of many small users (e.g. users requiring less than 5% of the total bandwidth) static pricing was *nearly optimal* (within 2% of the optimal static pricing policy) within the class of dynamic pricing policies. We believe that this was due to the way they modeled demand as a time-invariant, known rate of request of arrivals that varied only by price. While the [PT99] model allows an elegant structural analysis, the actual demand function would vary greatly in time – and most likely we would not be able to ascertain all of the factors which influence this demand.

Therefore we will assume a stochastically changing demand that can be modeled as a hidden Markov chain.

The method we develop will provide efficient resource allocation, at least in terms of revenue maximization for the service provider. That is, we define 'efficient' in a monopolistic sense in that those who are willing to pay the highest price for a service must value it most, and therefore will receive the service. By 'guaranteed services' we mean that if a user is allowed access he or she will have reserved for his use the required bandwidth for the time period required to complete the requested action. The method we will use in attempting to control the congestion is similar to congestion pricing, although in this context the term demand-dependent pricing is more accurate (since admitted customers will be guaranteed to get what they pay for there will be no congestion on the system). To do this we will develop a demand dependent pricing heuristic which will allocate resources in a broadband network, based upon partial observations of demand.

In addition, we will make the assumption that a service class can be categorized by its required 'effective bandwidth' [KG99] and we will make admission control decisions based upon this assumption. We will also assume that potential users can be informed of price changes instantaneously, so that they will not be able to order at a price that is not currently in effect.

1.3 Research Contributions and Objectives

The technical contributions we will make are (1) to generalize the model of [PT99] to the case where the demand functions are unknown and changing and (2) to determine whether demand uncertainty of this type widens the gap between the performance of optimal fixed (static) prices and optimal dynamic pricing policies. We suspect that an increase in revenue can be achieved by utilizing dynamic congestion tolls over the use of a static pricing policy when the demand functions are imperfectly observed and changing in time.

Our objective is to develop a methodology which can then be implemented to allow the use of inelastic services over a resource-constrained data network. In addition, the heuristic has broader applicability and could be converted to other domains where demand modeling is similarly difficult and resource constraints exist.

1.4 Thesis Organization

This thesis is organized into 8 chapters. Chapter 2 will discuss the background of this problem, to include

- the causes and current remedies of network congestion,
- pricing mechanisms for network and similar services,
- dynamic programming theory for Markov Decision Processes (MDPs) and Partially Observed Markov Decision Processes (POMDPs),
- the work of [PT99], from which this work is inspired

Chapter 3 will expand the model developed by [PT99] by including our uncertain, time-varying demand model and converting their formulation to fit the POMDP. Chapter 4 will explain the heuristic we have developed which provides a sub-optimal solution to the POMDP. We will discuss the computer code we developed to solve the problem and make some observations upon the data we have collected. We will also discuss the classification rule we used to estimate hidden demand states. Chapter 5 will show the development and solution technique for optimal static prices. Chapter 6 will analyze the differences between the optimal static and optimal dynamic prices for the case of known demand. In addition we will show comparison of these methods to the simulation we will later build in order to verify the calculations. Chapter 7 will explain the methodology that we used to compare the heuristic to an optimal static priced system for the case of unknown demand. Here we will provide the results of the comparison and a discussion of those results. Chapter 8 will present our conclusions and some recommendations for future work. Following Chapter 8 we include the Bibliography, and in the appendices the computer code we developed.

Chapter 2. Background and Related Research

In order to achieve our objectives it has been necessary to conduct research in several areas. We will begin by discussing network congestion and the techniques by which congestion is typically managed. Next, we will look at current pricing techniques for network services. Following this we will discuss dynamic programming for Markov Decision Processes (MDPs) and at the recent work of [PT99].

2.1 Congestion and QoS on Networks

Congestion results in a network when there is more load than can be adequately handled by the capacity of the network. An increase in congestion causes the Quality-of-Service (QoS) offered by the network, as measured in terms of transit delay (latency) and the variance in that delay (jitter), to be degraded. In the Internet, the TCP/IP protocol handles the administrative duty of congestion control by dynamically allocating bandwidth and deleting data when necessary [MSMO97] – both of which obviously affect the QoS. This results in the loss of time to network users and has resulted in a desire for increased bandwidth and QoS guarantees.

Expansion of capacity can be affected in several ways: increasing current transmission lines and associated hardware, enhancements to the physical layer technology that transmits the data, or advances in network or transport layer protocols. We will examine each of these in turn.

Increasing Capacity: Increasing the capacity of the physical layer (i.e. transmission lines and associated hardware) is a fast and easy, though costly, solution. Unfortunately it is also not an effective solution for very long. The "throw more bandwidth at the problem" solution is analogous to adding a lane for congested vehicular traffic; the extra lane only causes more people to feel that there are sufficient resources for traffic, so that new traffic as well as the traffic suppressed by the previous congestion is compounded, until the system is again in a congested state. Work by [GC98] has shown that the effect of adding capacity alone to a congested data network system is to increase the traffic to a point where the congestion and associated social costs are higher than they were initially. This would then require additional bandwidth in a never-ending cycle. This type of result is familiar in economic theory and is referred to as a 'tragedy of the commons' [MB99, GSW95] where the 'common' good (bandwidth) is abused by everyone because users do not pay the total cost for its use. Thus capacity expansion alone can be seen as only a temporary solution.

Increasing technology: Work is constantly being done to improve the physical layer of networks to make the transit of data faster, and thus to mitigate congestion. The cost of such technology is falling also – you can buy 10 times the bandwidth today for 1/10 the price of 5 years ago [CP99]. Unfortunately, although the cost of network technology is constantly falling, the increasingly high speed at which data can be transmitted is not keeping pace with the increases in demand [CP99, GC98, KA98]. Perhaps the biggest issue in this area is that the main component to the cost of increasing bandwidth to home users is the replacement of the 'local loops' that physically connect homes to the nearest router. This cost is high not because of the cost of the new lines but because of the digging to replace or put in these new lines [T96]. Without replacing these lines the benefits from other technological increases cannot be fully realized. Because of this the increases in technology are not always a cost-effective solution to the problem of needing to increase bandwidth.

Improving Protocols: Decreasing load on congested lines through improvement of transport techniques, or protocols, is another area where many advances have been made. Routing of data around congestion is an effective way of dealing with congestion in a large network [T96]. This method works by dynamically computing the best route through a network by utilizing current measures of the transit delay, throughput rate, or number of hops. One effect of this is that the route the traffic takes to avoid congestion is generally slower than the optimal route. This is irrelevant to elastic traffic, such as FTP, but for delay sensitive traffic, such as telephony, this can have a significant impact. For inelastic traffic not only the transit delay but also the jitter (the variance of the transit delay) must be minimized. An application such as a VTC is significantly degraded when the route is dynamically changed often – thus this solution is not effective for all applications. With the use of a connection oriented network technology such as ATM, routing can be effective in ensuring QoS to applications that have already been 'connected', but cannot help new users when congestion is already present (because they cannot obtain service which will support their QoS requirements). Other technological improvements in traffic protocols, such as flow shaping, also suffer from similar problems. Flow shaping controls the rate at which traffic is transmitted to ensure a constant flow [T96] – it does this by buffering data and releasing it at a certain rate. This is adequate if only minimizing jitter is required, but if minimizing transit delay is imperative then this technique may only compound the problem. The complete solution to the problem of eliminating congestion, therefore, is not an increase in bandwidth through either physical or technological methods.

QoS guarantees are commitments by a network provider to deliver some standard of service – such as reliability, throughput rate, transit delay, or jitter. The ability of a network to meet this standard is dependent upon the available resources. QoS guarantees with respect to transit delay and jitter are normally pursued through previously discussed methods, or through flow control and admission control. Flow control is the regulation of the speed at which the sending device transmits data. This method only works when there is a large tolerance in transit delay. Admission control, on the other hand, only allows traffic flow when there are available resources to support the QoS guarantee. To utilize admission control two things are necessary, (1) available resources and (2) user demand for the service. From this concept springs forth the idea of congestion-based pricing of network services - where the cost of using bandwidth is based upon the congestion currently experienced in the system [BRS99, GC98, GSW95, GSW95d, KA98, KG99, KM99, KMB99, PT98]. *Instead of changing the system to meet the demand, this technique attempts to change the demand to meet the system.* Utilizing this type of pricing scheme ensures that only users willing to pay the actual cost (to include additional cost induced by congestion) actually use the network – thus avoiding a tragedy of the commons. Before we examine this 'congestion pricing' in detail, we will review some of the more common pricing scenarios for Internet/network service, to see why they do not efficiently deal with congestion.

2.2 Traditional Pricing Techniques

Typical Internet Service Provider (ISP) pricing strategies today allow a user unlimited access for a fixed monthly fee of $10 - $25. Once the monthly fee is paid there is no reason for the consumer to stop accessing the service – indeed this type of pricing scheme has been shown to increase the amount of use artificially (people feel they are getting more for their money if they use the service more) [GC98]. Because of this, the fixed amount per month pricing scheme, although currently popular, has been shown to be inefficient. A current happenstance in the field is increased pricing for services that offer higher bandwidth – such as DSL or cable network connections. These types of networks do not necessarily guarantee higher throughput rates, but are capable of providing this decreased transit delay for larger amounts of data.

Interestingly, it used to be the case that many ISP's would have a fixed fee per month for access, plus a charge proportional to access time, providing an incentive to eliminate idle access time. Unfortunately this type of pricing scheme is also inefficient. Consider, for example, a user who logs onto an ISP, read his email for 10-20 minutes, responds to several of those emails, and then logs off after 45 minutes. The user had access for 45 minutes, but was actually only sending or receiving data for several seconds. Thus, again, we have an inefficient system, because the person who logs on for 45 minutes and utilizes every second of it to download a new demo game would pay the same amount as the user for email did. With the rising popularity of America On-Line (AOL), which was among the first to charge a fixed amount per month for unlimited access, this type of pricing scheme became very unpopular and has since disappeared.

Many businesses utilize a method of purchasing blocks of bandwidth from network providers. These purchases can be in the form of leased lines or by utilizing virtual leased lines [CP99]. Because the business has the capability of specifying what traffic is allowed to go onto a link from its end, this technique can be effective in reducing congestion (or even eliminating it), but some portion of time the network would not be at it's optimal load, and thus there is a form of 'waste'. The network owner could have sold this available resource and increased his revenue or the leasee could have either ‘re-sold’ this excess or could have reduced the amount of bandwidth it had contracted for, thus reducing costs.

Recently several ISP’s have offered completely free connection service. The ISP is able to do this because of advertising monies paid to it by companies to put ads on their websites, and subsequently upon the desktops of the subscribers. While this method is likely to become popular with consumers who do not desire tight QoS constraints, we feel that this type of provider will not be able to offer the services we are interested in due to the cost of the enhanced hardware, and so while this ISP may be useful for email or limited WWW and file swapping, it will not be effective in providing inelastic service.

2.3 Congestion Pricing

In congestion pricing users are charged based upon both the amount of traffic they produce and the ambient congestion that is currently experienced in the network. Typically, the charge increases with ambient congestion in an attempt to influence the demand. There are several objectives that have been presented when utilizing a congestion-pricing scenario, namely the maximization of some form of network utility. This utility has been defined in various ways - as the revenue of the network provider, as the overall benefit to the users, as the total throughput rate of traffic. In the context of broadband QoS networks, congestion pricing works by defining various service classes (with diverse QoS guarantees) and then charging an amount for each service based on the current state of the network. In this research we will assume that QoS can be delivered by dividing up effective bandwidth [KG99] to customers in the amount they desire and charging them some fee for its exclusive use, similar to the work done by [PT99].

Congestion pricing is a specialized form of dynamic pricing where the price is set by the amount of congestion on a system. In more general terms dynamic pricing is simply a technique where prices change at certain times, for various reasons. In our society that reason is mainly because more revenue can be made by changing to a different price for an item, often based upon the elasticity of demand - though the case does exist, in vehicular traffic, where the objective is simply the reduction of congestion. While people enjoy the benefits of this technique – when prices are down – they are usually rather unhappy when prices rise. Dynamic pricing is used extensively in some industries – such as airlines (yield management), automobiles (prices generally changed through rebates and dealer incentives), and in the hotel industry. Another familiar example of dynamic pricing is the long distance telephony industry. This industry has for years varied price by time of day in an effort to increase revenues. In general, these dynamic prices are set based upon the demand for the product. When demand exceeds supply prices can be raised – and revenues increased. This price change, of course, affects the demand. We believe that the long distance pricing methods also had the effect of helping to control demand on the resources. It is precisely this aspect of dynamic pricing we are interested in capitalizing upon. Interestingly, some long-distance companies are currently introducing static prices. We believe this is caused by an increase in available resources allowing an increased revenue to be generated because it is similar to a situation of unlimited resources. We will discuss the unlimited resource situation a little later when we cover the results from [PT99].

Pepsi-Cola was recently assailed in the media for working on what was claimed to be technology that would allow vending machines to adjust prices for soda based upon the ambient air temperature – so a cold soda would cost more in hot weather, when people are thirstier. In response Pepsi-Cola announced that it was working on technology that would allow it to *reduce* the cost of soda at certain times in an attempt to increase sales, and that technology was being developed to allow the company to monitor the status of its machines on-line (status as being the amount of inventory on hand). This reply seemed to appease the media, but it brings to light the very volatile attitudes towards congestion or demand-dependent pricing. It seems that it is acceptable to reduce prices at times of low demand (which also increases revenues), however it is unacceptable to raise prices when demand is high. So companies can simply mark up their products and then 'discount' them at an appropriate amount – no one raises a stir when a rebate ends, especially if the rebate is only adjusted to a slightly lower amount. It is our contention that there is actually no difference in the methodology, and that it is still demand pricing, but we understand that there is a psychological effect on customers.

2.4 Markov Decision Processes and Dynamic Programming

Our model will take the form of an imperfectly observed, continuous time, infinite horizon, average reward dynamic programming problem. This type of problem is complicated not only because is it an infinite horizon formulation, but also we must deal with imperfect information about the demand for network services. In this section we will develop formulations of this problem starting from a simpler model and increasing in complexity till we reach the model we will use.

2.4.1 Discrete-time, Finite-horizon Dynamic Programming Models and MDPs

Markov Decision Problems are problems that attempt to optimize a system by applying controls at appropriate times. The control can be applied at the beginning of a *stage*. A stage is a period of time where the system is in some *system state,* which can be represented by state variables. The Markov Assumption states that a system can be completely defined by the state variable in the stage that it is in, with no knowledge of past states. We can then apply a control (change some parameter of the system which is under our control) and the current state variables along, with that control, can completely predict the value of the state variables at the end of the next stage. This is precisely why it is a *Decision* model – we wish to make an optimal decision to control the system. Dynamic programming as discussed is one of the main techniques used in solving this system to find the optimal controls.

Dynamic programming problems are optimization problems where the objective function is the expected value of the sum of rewards accrued at each stage of a sequential decision process. They are solved in general by first computing the optimal reward-to-go from the point in time one stage away from termination. Next, the optimal reward-to-go from two stages to go is computed, and so on, until the optimal reward to go from the current time period is obtained. Along the way the actions that are optimal for state/stage pair are stored to yield the optimal policy for maximizing expected revenue.

Many problems have a fixed number of stages (N) with a terminal cost (revenue) at the end of the last stage. If the revenue at each stage can be described by the following function:

$$g(x,u,w)$$

where x summarizes the state of the system at a time k stages remaining, u is a control applied at this stage, and w is a random parameter, then the optimization model yields the following recursion:

$$J_{k+1}(x) = \max_{u \in U(x)} \underset{w}{E}\{g(x,u,w) + J_k(f(x,u,w))\} \qquad k = 0,1,\ldots,N$$

and the optimal revenue $J_N(x)$ can be generated after N iterations.

Most problems of this type are solvable in closed form, as long as the number of possible states at each stage is not too large. This type of formulation is used to solve problems such as inventory control or machine replacement. The size of 'too large to solve' in this case is generally a very small number because the difficulty of the problem grows exponentially with each stage. Thus a one-stage problem with 10 states has only 1^{10}, or 10, calculations. The same problem with 2 stages has 10^2, or 100 calculations. Now if we expand that to a 10-stage problem with 10 states we must consider 10^{10}, or 10 billion calculations. The problem we are considering in this thesis will later be shown to have an uncountably infinite number of states. Also, the formulation has an infinite horizon (number of stages). The reason for this is that we do not want to simply generate a pricing solution for the next week, or next month, or year – we want a solution we can use indefinitely. While it may not be correct to assume that this problem will exist with it's current parameters forever, we seek a solution that will allow us to generate near optimal gains for the foreseeable future, and so the infinite horizon aspect of the model seems appropriate.

2.4.2 Discrete-time, Infinite-horizon Models with an Average-Reward Criterion

Thus far we have developed this formulation on the basis of discounting or because of the existence of a reward free terminating stage which we would eventually arrive at; in the actual formulation of this thesis neither of these is the case. In this situation it is beneficial to optimize the average cost per stage.

For the infinite horizon, average cost model we want to maximize the limit at any stage i shown here:

$$\lim_{N\to\infty}\frac{1}{N}E\left\{\sum_{k=0}^{N-1} g(x_k,\mu_k(x_k)) \mid x_0=i\right\}$$

where $\mu_k(X_k)$ is an element from the control space. If the decision process is irreducible, that is if all states communicate under all policies $\pi = \{\pi_0,\pi_1,...\}$, then the optimal average reward from each state is independent of i. This is because the above limit considers all possible states, and as the number of stages approaches infinity any effect that the initial state could have had on the system is averaged to be of no more significance than any other state which the system passes through along the way. Stated more succinctly, the contribution to the total reward from the early stages becomes insignificant as N approaches infinity, or

$$\lim_{N\to\infty}\frac{1}{N}E\left\{\sum_{k=0}^{K} g(x_k,\mu_k(x_k))\right\}=0. \quad \text{for any fixed K}$$

The following proposition is taken from [B95]:

Under these assumptions the following hold for the average cost per stage problem:

(a) The optimal average reward λ^* is the same for all initial states and together with some vector h* = [h*(1),...,h*(n)} satisfies Bellman's equation

$$\lambda^* + h^*(i) = \max_{u \in U(i)} \left[g(i,u) + \sum_{j=1}^{n} p_{ij}(u) h^*(j) \right], \mathrm{i} = 1,\ldots,\mathrm{n}$$

Furthermore, if μ(i) attains the maximum in the above equation for all i, the stationary policy μ is optimal. In addition, out of all vectors h* satisfying this equstion, there is a unique vector for which h*(n)=0.

(b) If a scalar λ and a vector h = [h(1),...,h(n)} satisfy Bellman's equation, then λ is the average optimal revenue per stage for each initial state.

For a detailed proof of this proposition see [B95].

2.4.3 Continuous-Time, Average-Reward Decision Processes and Uniformization

Up until this time we have discussed processes where rewards are accrued only at certain times – thus the *discrete-time* aspect. In some situations rewards can accrue at any time in a continuous spectrum – for example at a random time an arrival to a queue could occur, and that could be the catalyst for the generation of rewards. The problem this thesis addresses is such a type, where arrivals to the system occur according to a Poisson process, and upon arrival we receive a reward and then set the control which will control the next stage.

This type of formulation is more difficult to develop because instead of summing rewards as they occur at discrete intervals we must calculate integrals over the time periods:

$$\lim_{N\to\infty}\frac{1}{N}E\left\{\int_0^T g(x_k,\mu_k(x_k))\right\}$$

In order to calculate this value it is possible to discretize this continuous function by considering as the time interval the shortest possible transition time of the model. To be able to do this we simply must be able to calculate the maximum transition rate to predict with probability p_{ij} the state of the system after applying control $\mu_k(x_k)$. Take, for example, the following model, where the transitions rates are shown:

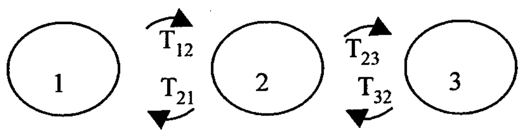

Figure 2.1 Markov Chain

If all transition rates are equal then the maximum rate is simply is simply T, shown above, but if the rates are not equal and/or are dependent upon the state the system is in the calculation becomes more complicated. In this case we must add a self transition to the model, so that the transitions occur at the same rate from each state (but that transition may be a transition to itself):

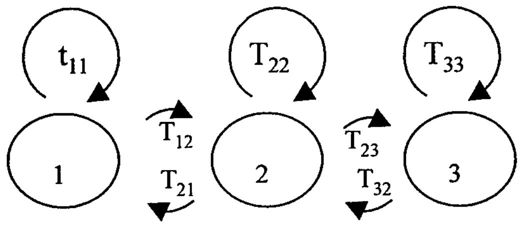

Figure 2.2 Markov Chain with self-transitions

To calculate the maximum transition rate we sum the maximum increasing rate (increasing in terms of state variable) with the maximum decreasing rate. We can then change the probabilities of transition by dividing the actual transition rate for a state (increasing or decreasing) by this maximum rate:

$$P'_{i,j} = (T_{i,j}/\nu) * P_{i,j}$$

At this point we can resume the process of solving the problem as if we were working with a discrete-time problem. We can then formulate a Bellman's equation of the form:

$$J(i) = \min_{u \in U(i)} [\sum_j p'_{ij}(u) J(j)]$$

2.5 Congestion Pricing of Network Services: The [PT99] Model

As was mentioned earlier, this thesis will primarily carry forward research conducted by [PT99], who modeled a single network link which offers M different broadband service classes. They assumed that the network has some amount of resource (bandwidth) R, and that each active user of class i needs some amount of that resource, r_i. The vector of all required resources is $\mathbf{r} = (r_1,\dots,r_M)$. The value of r_i is the effective bandwidth [KG99] required by the i^{th} service class. Each class i will consume bandwidth for an exponentially distributed service time with a rate of μ_i. Upon arrival the user pays a fee, u_i, set by the service provider. Let $\mathbf{u} = (u_1 \dots u_M)$. It is assumed that a known demand function exists which relates the request arrival rate to the price of service for that class:

$$\lambda_i = a_i + b_i u_i$$

where a_i and b_i are known. It is also assumed that there exists a price, $u_{i,max}$, beyond which the demand becomes equal to zero, and that the function is continuous and strictly decreasing in the range of $(0, u_{i,max})$. Note that **u** is set by the service provider and is allowed to vary in response to the state of the network.

The vector of demand rates can be written as

$$\lambda(u) = (\lambda_1(u_1), \ldots, \lambda_M(u_M))'$$

with the arrival rate when all prices are equal to zero denoted by λ_0. Because of monotonicity $\lambda_0 \geq \lambda(u)$ for all non-negative price vectors **u**. λ_0 represents the upper bound for the maximum arrival rate, and $\lambda_{i,max}=\lambda(u_{max})=0$ represents the lower bound. This model can be represented by the following figure:

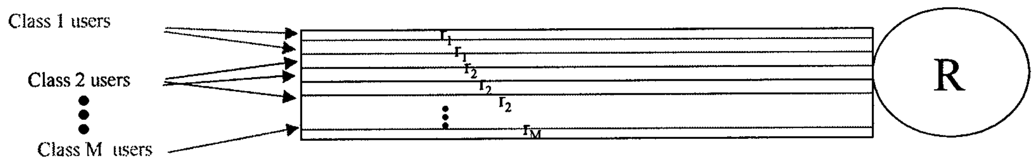

Figure 2.3 Broadband Network Model

The state of the system is by characterized the number of users of each class in the system at a particular time. We define $n_i(t)$ as the number of class i users in the system at time t, so that the system's state can be represented as

$$N(t) = (n_i(t), \ldots, n_M(t))'$$

Because $n_i(t)$ is discontinuous at times of call arrivals, $n_i(t)$ is treated (as a convention) as a right-continuous function of t.

An incoming call from service class i at time t is accepted if and only if available resources are available to service that call, i.e. whenever $N(t)'\mathbf{r}+r_i>R$. Otherwise, the call is 'blocked', or not allowed service. We can enforce this rule by setting $u_i = u_{i,max}$, and this will result in no class i arrivals.

At this time we will formally define what we mean by a *policy.* A policy is a mapping from $N=(n_1, \ldots, n_M)$ to prices for class 1, ..., class M service. The prices are represented as

$$\text{Price for Class } i = (\Pi_i(N, t))$$

Note that this mapping is dependent upon both the system state, N, and the current time, t. For a stationary policy the mapping does not depend explicitly upon time, thus

$$\text{Price for Class } i = (\Pi_i(N))$$

A Static Policy is a policy where for all states the control is a constant value:

$$\Pi_i(N) = \Pi_i(\overline{N}) \text{ for all } N, \overline{N}$$

while in a dynamic policy

$$\Pi_i(N) \neq \Pi_i(\overline{N}) \text{ for some } N, \overline{N}$$

For this model we defined the system state as $\mathbf{N}(t)=\{n_1(t)....n_M(t)\}$, and the policy can be defined as $\Pi(N(t)) = (\pi_1(N(t),,\ldots, \pi_m(N(t))'$. The price vector is $\underline{U}= (\pi_1(N(t),,\ldots, \pi_m(N(t))'$, and the arrival rate as a function of state can be written as $\lambda(\Pi(N(t)))= (\lambda_1(\pi(N(t))) ,\ldots, \lambda_m(\pi(N(t))))'$, which is equivalent to $(\lambda_1(u_1) ,\ldots, \lambda_m(u_m))'$.

This system evolves as a continuous time Markov Chain. Given the current time t and price $\mathbf{u}(t)$, and a small change in that time, δ, it is possible to estimate the probability of a class i arrival in the next δ time units as $\lambda_i(u_i(t))\delta = \lambda_i(\pi_i(N(t)))\delta$. The expected class i revenue generated from this time δ would be approximately equal to

$\delta u_i(t)\lambda_i(u_i(t)) = \delta(\pi_i(N(t)))\lambda_i(\pi_i(N(t)))$. As δ decreases these estimates becomes more accurate. The long-term average reward can therefore be written as

$$J_\Pi = \lim_{T\to\infty}\frac{1}{T}E[\int_o^T \lambda(\Pi(N(t)))'\Pi(N(t)dt]$$

The objective of [PT99] was then to identify the pricing vector **u** as a function of the state of the network that maximizes the value of the above quantity.

Note that all states in the formulation of this problem communicate, and that the demand functions $\lambda_i(u_i)$ are continuous. Because of this the transition rates as well as the reward rate are continuous in the decision variables. Both the reward rate and the expected holding rate at each state **N** is a bounded function of **u**, as is the total transition rate out of any state

$$\upsilon = \sum_{i=1}^{M}(\lambda_{0,i} + \mu_i[R/r_i]).$$

This value of ν is the greatest possible number of transitions per time period, an utilizing it we can unioformize this Markov chain, as discussed earlier. We can thenm formulate a Bellman's equation:

$$J^* + h(N) = \max_{u\in u}\Big[\sum_{i\notin C(N)}\lambda_i(u_i)u_i + \sum_{i\notin C(N)}\frac{\lambda_i(u_i)}{\upsilon}h(N+e_i) +$$
$$\sum_{i=1}\frac{n_i\mu_i}{\upsilon}h(N-e_i) + (1 - \sum_{i\notin C(N)}\frac{\lambda_i(u_i)}{\upsilon} - \sum_{i=1}\frac{n_i\mu_i}{\upsilon})h(N)\Big]$$

for all N

where $$e_i = (0,\ldots,0,1,0,\ldots,0)',$$

$$C(N) = \{i \mid (N+e_i)'r_i > R\}$$

is the set of classes whose calls cannot be admitted in state N. Following is a breakdown of the elements of this equation:

$$J^* + h(N) = \max_{u\in U}\Big[$$

Optimal Average Revenue

Relative Revenue at State N

$$\sum_{i\notin C(N)} \lambda_i(u_i)u_i$$

Instantaneous Revenue Rate

$$+ \sum_{i\notin C(N)} \frac{\lambda_i(u_i)}{\upsilon} h(N+e_i)$$

Pr (user i arrival) * (Relative Revenue at State N+e$_i$)

$$+ \sum_{i=1}^{M} \frac{n_i\mu_i}{\upsilon} h(N-e_i)$$

Pr (user i departure) * (Relative Revenue at State N-e$_i$)

$$+ \Big(1 - \sum_{i\notin C(N)} \frac{\lambda_i(u_i)}{\upsilon} - \sum_{i=1}^{M} \frac{n_i\mu_i}{\upsilon}\Big) h(N)\Big]$$

Pr (remaining in this state) * (Relative Revenue at this State)

Note that we actually have a system of equations since the equation for h(N) depends upon h(N+e$_i$). By imposing the condition that h(**0**)=0, this Bellman's equation has a unique solution. This is due to the fact that for any two states, N$_1$ and N$_2$, there exists a stationary policy that makes N$_2$ reachable from N$_1$, i.e. the fact that all states communicate. Once this equation is solved an optimal price vector can be obtained by choosing the price vector **u** at each state **N** that maximizes the right hand side of the equation. The solution to this equation can be obtained by utilizing classical dynamic programming algorithms, however complexity increases with the size of the state space.

Because of this an exact solution is only feasible when the number of classes is small. We will explain in more detail the algorithm used to solve this system of equations in section 4.2.

[PT99] obtained the following theoretical results about their model:

- In the case of unlimited capacity $(R=\infty)$ the optimal price vector is a constant (u_{inf}). (Thus in this situation the optimal dynamic pricing policy is a static pricing policy).
- In a single class system the optimal price is an increasing function of the number of users already admitted in the system.
- For the asymptotic case of many small users, near-optimality results from static prices within the class of dynamic pricing policies.

In this last result, we define small by examining the system as the percentage of the total bandwidth absorbed by any one user approaches zero. In other words, we examine the average revenue in the following situation, as $\gamma \to \infty$:

$$R_\gamma=\gamma R$$
$$\lambda_{i,\gamma}=\gamma\lambda_i(u)$$
$$\mu_i \text{ constant}$$
$$r_i \text{ constant}$$

For us the most interesting aspect of this earlier work by Pachalidis and Tsistiklis is the analysis of this small users asymptote, which seems to indicate that dynamic pricing may not play much of a role in broadband networks with high levels of aggregation. The main objective of this thesis is to determine whether a similar result holds when we relax the assumption of a known, time-invariant demand.

Unfortunately for us, real life problems are often much more complex than this. In our case we will not be able to obtain complete observations of the state variables, therefore we will have to deal with *partial observations*. This partial observation is due to a stochastic process which does not provide completely accurate feedback – just a request for service at certain time intervals. In more generic terms this type of problem provides some type of imperfect state information which can be used to attempt to optimize the problem, though the solution is generally more intensive computationally. Because of this these type of problems are often solved sub-optimally.

One of the main techniques used to solve POMDPs sub-optimally is called Open Loop Feedback Control [B95]. In this technique a decision is made about the partially observed system on the basis that the most recent observation will be the last observation received. When another observation is subsequently received it is then considered and another decision is made. We will later use this method in the development of our heuristic.

Chapter 3. Problem Formulation: The Case of Partially Observed Demand

The pricing model analyzed in [PT99] is highly structured and may be interpreted as a priced multiclass loss model. While this model structure admits a very clean analysis, leading ultimately to a proof of the near-optimality of optimal static pricing, this comes at the expense of several rather strong simplifying assumptions. As is usually the case in multiclass loss models, it is assumed that requests for various types of service arrive according to a price-dependent Poisson process. The instantaneous rate of request arrivals for each service class i is a known function $\lambda_i(u_i)$ of the price u_i set for that service. The functions λ_i parameterize the problem formulation, and the ISP is allowed to exploit this knowledge in optimizing its revenue. Inherent in this is the assumption that after changing the price u_i for a service class the effect of the change is instantaneous, i.e. that the new price is communicated instantaneously to the pool of potential network users. Both of these assumptions, (1) known, time-invariant demand and (2) instantaneous communication of service prices, are somewhat unrealistic in the context of broadband services. In this thesis, we intend to relax the former. Our hypothesis is that in modeling demand as an unknown, stochastically varying quantity the difference between optimal dynamic and optimal static pricing is significant even in the asymptotic case of many small users.

In the generalized model we are about to present, we model demand as a stochastic process which is observed only through the arrival of requests at varying rates.

The service provider must estimate demand from the number of requests for service as a function of price, and then utilize that observation to control the system (by setting prices) and influence its behavior towards a more profitable state. From this interarrival time observation we will compute the posterior probability distribution that the service class is in a particular demand state. From this probability distribution and from the other state variables (numbers of current users per service class), we set the price for services which will drive the system toward increased revenue. In Section 3.1 below, we present the new partially observed, stochastically varying demand model. In Section 3.2, we use this model in formulating the problem as a partially observed Markov decision process. In Section 3.3, we illustrate the state-space complexity of the new formulation.

3.1 A Model for Partially Observed, Stochastically Varying Demand

As a first cut at modeling unknown demand, we have chosen to represent our uncertainty as a hidden Markov chain. We assume that at any point in time the instantaneous rate of arrivals within a service class i can be represented as a linear function of the service price u_i. The parameters of this line are determined by the state of the hidden Markov chain. This is a generic model for demand for which we have yet to develop practical statistical tools for parameter estimation. For the computational experiments which appear later in this thesis, we set the model parameters arbitrarily.

3.1.1 The Hidden Markov Chain

In our model, the rate of arrivals is determined by the state of a hidden Markov chain. The state of the Markov chain selects the linear function that relates price to the instantaneous rate of arrivals. For our computational study, we assume that the Markov chain is a three-state birth-death process in such a way that

1. each state of the Markov chain can be interpreted roughly as a demand level, e.g. "low," "medium," and "high, and
2. the demand state for a service class cannot jump all the way from "low" demand to "high" demand and vice versa, without passing through "medium" along the way.

The model for service class i is illustrated in the figure below.

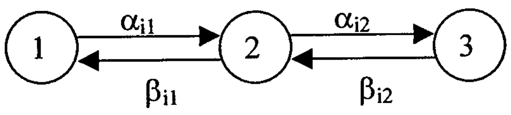

Figure 3.1 Demand-State Modeling

States 1, 2, and 3 in the figure correspond roughly to "low," "medium," and "high," as discussed in the following subsection. The labels on the arcs in the figure are defined as follows:

1. α_{ij} is the average rate of transitions of class i from demand state j to demand state j+1.
2. β_{ij} is the average rate of transitions of class i from demand state j+1 to demand state j.

A similar model structure is used for each service class $i = 1, \ldots, M$.

We point out again that the demand state for each service class is not directly observed by the service provider. Information about the demand state of a given service class comes only by recording a history of arrivals and reconciling the average rate of arrivals with the prevailing set of prices for service.

3.1.2 The Rate-Price Relationship for a Given Demand State

As mentioned above, we assume that if the demand state of a service class is constant, then the instantaneous rate of request arrivals depends linearly on the price. We refer to the rate-price relationship for a given service class *i* and demand state *c* as a *demand function*, $\lambda^c_i(u_i)$, and each such function reflects the fact that as the price u_i increases the instantaneous rate of arrivals decreases. Thus, each demand function is of the form $\lambda^c_i(u_i) = a^c_i + b^c_i u_i$, where a^c_i and $b^c_i < 0$ are parameters associated with service class *i* and demand state *c*. For the computational study which appears later in this thesis, we will assume that the demand functions for a given service class do not cross, as shown in the figure below.

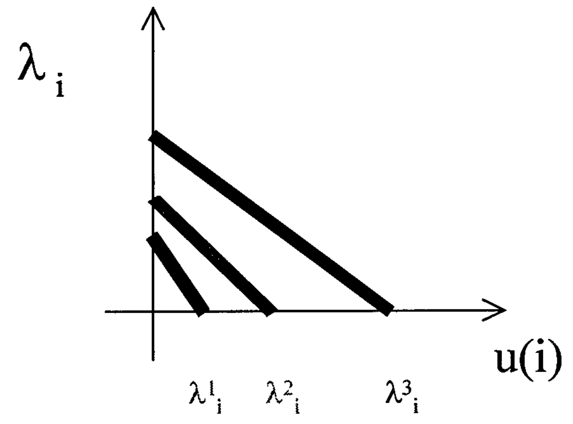

Figure 3.2 Demand –State Illustration

3.2 Pricing as a Partially Observed Markov Decision Process.

The pricing model considered in [PT99], with known and time-invariant demand functions, reduces to a continuous time Markov decision process. In our case, since the demand function for a given service class is selected according to the state of a hidden

Markov chain, the pricing model becomes a partially observed Markov decision process (POMDP).

If the demand states for each service class were observable, then the state of the decision process would be $N(t) = \{ [n_1(t), d_1(t)], \ldots., [n_M(t), d_M(t)] \}$, where $n_i(t)$ is the number of active class i sessions at time t, and $d_i(t)$ is the demand state for user class i. $N(t)$ is all that would be needed to determine the probability distributions that describe (1) the time to the next event and (2) whether the next event corresponded to a request arrival, user departure, or a demand state change. With perfect information about the demand, the decision process in this case would be very similar to the model discussed in [PT99], a continuous-time MDP.

Unfortunately, the "demand state" that characterizes the demand for a type of service from a large pool of users is an abstract concept that lacks a real physical interpretation and cannot be directly observed. The best we can do is to maintain, as a sufficient statistic, the probability distribution ρ_i for the demand state of each user class i. Thus, the state of the partially observed decision process is $\boldsymbol{S}(t) = \{ [n_1(t), \rho_1(t)], \ldots, [n_M(t), \rho_M(t)] \}$. In principle, we want to identify a stationary policy $\Pi(\mathbf{S}) = [\ \pi_1(\boldsymbol{S}), \ldots, \pi_M(\boldsymbol{S})\ ]'$ that maximizes long-term average revenue for the service provider

$$\lim_{T\to\infty}\frac{1}{T}E\left\{\int_0^T\left[\Lambda[\Pi(S(t)),D(t)]'\,\Pi(S(t))\right]dt\right\}$$

where $\pi_i(\boldsymbol{S})$ is the price that the policy Π dictates for service class i when the system is in state $\boldsymbol{S}$, $\boldsymbol{D} = \{d_1, \ldots, d_M\}$ is the profile of hidden demand states, $\Lambda[\Pi(\boldsymbol{S}), \boldsymbol{D}] = [\lambda^{d1}{}_1(\pi_1(\boldsymbol{S})), \ldots, \lambda^{dM}{}_M(\pi_M(\boldsymbol{S}))]'$ is the vector of request arrival rates associated with the policy Π and demand state $\boldsymbol{D}$.

For the computational study that appears later in this thesis, we focus on problems with two service classes where the demand for each service class is characterized by a three-state hidden Markov chain. In this case, the state of the POMDP can be expressed as $S(t) = \{ n_1(t), \rho^1{}_1(t), \rho^2{}_1(t), \rho^3{}_1(t);\ n_2(t), \rho^1{}_2(t), \rho^2{}_2(t), \rho^3{}_2(t) \}$, where $\rho^c{}_i(t)$ is the probability at time *t* that service class *i* is in demand state *c*.

At this point, we note that our formulation of the pricing problem as a POMDP leads to a huge increase in the complexity of computing solutions. Now that the state ***S*** of the process involves the posterior distribution ρ_i for each service class *i*, the state space of the decision process is uncountably infinite. (This is because the space of all probability distributions is a continuum.) The state space complexity of our formulation causes significant difficulty in computing optimal pricing solutions. We explore these issues in the next subsection.

3.3 State Space Illustration / Complexity Issues

Let us first examine the number of states in the [PT99] formulation in order to have a comparison. Again, we are considering this model in accordance with the assumption that the amount of bandwidth required for an application can be estimated by its effective bandwidth. In this manner it is possible to make admission decisions on the basis of the total number of users in the system. As an illustration, we will examine a simple model of the system:

$M = 2$ (number of service classes)

$R = 10$ (amount of resource)

$r_1 = 3$ (amount of resource required by service class # 1 (i.e. effective bandwidth))

$r_2 = 1$ (amount of resource required by service class # 2 (i.e. effective bandwidth))

[PT99] represented the state of their system with the following notation:

$$\mathbf{N} = (n_1(t), \ldots. n_M(t))$$

and so this system would be represented as $\mathbf{N} = (n_1(t), n_2(t))$. The state space is as shown below:

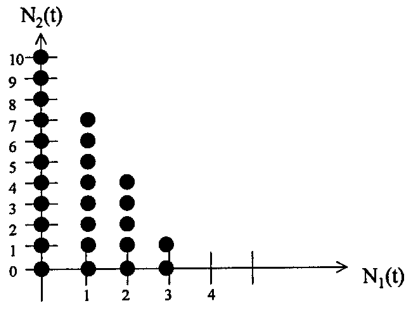

FIGURE 3-3 State Space Illustration

Note the discrete nature of the state space. The number of points in this state space is a function of the total available bandwidth and the requirements of each service class. For the simple example above the state space is defined by the constraint: $n_1 r_1 + n_2 r_2 \leq R$, i.e. $3\, n_1 + 1\, n_2 \leq 10$, which yields a state space of 26 points.

By expanding the model as we have to include demand states, we must not only consider the number of users in the system in each service class at time but also the actual values of the demand states for each service class. If this were a perfectly observed system, the state could be represented by $N(t) = [n_1(t), d_1(t), n_2(t), d_2(t)]'$. So, the number of states for this example grows by a factor of $C^M = 9$, where C is the number of possible demand states per service class(equal to 3 in this case). All totaled, we would have 26 * 9 = 234 possible system states, as opposed to 26 for the [PT99] model.

For the POMDP formulation with the state represented as

$$\mathbf{S(t)} = \{n1(t), \rho^1 i , \rho^2 i , \rho^3 i ;\ n_M(t), \rho^1_M , \rho^2_M , \rho^3_M \},$$

we have a similar number of values for the number of users in the system at time t, but the demand state parameters are now continuous, therefore we have an infinitely uncountable state space. One of the techniques often used when working with POMDPs is to discretize this state space in order to reduce the amount of states it is necessary to contend with. If we were to approximate the probability distribution of a service class being in a demand state to the nearest 10% we can estimate the number of states in the state space of this simple model as follows:

$$\# \text{ possible states} = 26 * 10^3 * 10^3 = 26{,}000{,}000 \text{ states}$$

While this is much simpler to work with then an infinitely uncountable number of states, it is still impossible to solve in closed form and exact numerical calculations are impractical for today's computers [B95], therefore we have developed a heuristic to solve the problem for a suboptimal solution.

Chapter 4. A Heuristic for the POMDP

In the preceding chapter we formulated the problem of revenue maximization for the service provider as a partially observed Markov decision process in continuous time. Since the state space of the formulation is uncountably infinite, there is very little hope for computing exact solutions for an infinite horizon (average cost) objective function. We postpone the exact characterization of optimality for this class of problems as future research. Our approach here is to develop an heuristic solution to the problem and then compare the value of this heuristic to the optimal static price which can be closely approximated through numerical calculations.

In Section 4.1 below we define the heuristic which we test later in Chapter 7. The heuristic involves consideration of all possible demand state profiles D = (d1, ..., dM), and then computing optimal stationary policies for each known, time-invariant demand case. At run time we continually compute the posterior probability distribution for the demand state associated with each user class, and then select the action specified by the optimal policy for the most likely demand state profile. In Section 4.2, we outline our methodology for computing solutions for the each known, time-invariant demand function profile. The same numerical technique could have been used by Paschalidis and Tsitsiklis in [PT99] in their computational examples. In Section 4.3, we outline our methodology for the on-line computation of the posterior probability distribution for the demand state of each user class. This is an integral part of the demand state estimate used in the heuristic.

4.1 Formal Definition of the Heuristic

The Heuristic we have developed to obtain improvement is as follows:

1. Identify the parameters of the demand model

2. Solve the associated MDP's (one MDP for each demand state pair)

3. Run time operation:

    a) Obtain demand state estimate (select the most likely demand state pair)

    b) Set price

    b) Monitor system for Arrival/Departure and revise demand state estimate

4. Repeat step 3.

    • Periodically restart to accommodate changes to the model parameters

4.2 Solutions to MDP's

The procedure we have used to solve the MDP's is a standard dynamic programming algorithm, known as relative value iteration [B95], designed to compute a solution to the Bellman's equation from Section 2.5. We implemented relative value iteration in the C programming language , and the following pseudocode highlights the main elements of the routine.

```
while convergence has not occurred
{
•  set largestConvergence = 0
•  for each possible state ($n_1$, $n_2$)
      {
      •  for every possible price for class 1 ($u_1$)$^@$
         {
         •  for every possible price for class 2 ($u_2$)$^@$
            {
            •  solve$^@$
```

```
                    $htemp = \sum_{i \notin C(N)} \lambda_i(u_i)u_i + \sum_{i \notin C(N)} \frac{\lambda_i(u_i)}{\upsilon} h(N+e_i) + \sum_{i=1}^{M} \frac{n_i\mu_i}{\upsilon} h(N-e_i) + (1 - \sum_{i \notin C(N)} \frac{\lambda_i(u_i)}{\upsilon} - \sum_{i=1}^{M} \frac{n_i\mu_i}{\upsilon}) h(N)$
                    • save as $h_{best}$ the highest value in this iteration
                    • save as $u_{opt}(1)$ and $u_{opt}(2)$ the prices which achieved the optimum for $n_1$ and $n_2$
                    • if $n_1 = 0$ and $n_2 = 0$, then set $h_{best0} = h_{temp}$
                  } /* end for $u_2$ */
              } /* end for $u_1$ */
            • set $h(n_1, n_2) = h_{best} - h_{best0}$
        • set convergence = $| h(n_1,n_2)_{\text{last iteration}} - h(n_1,n_2) |$
        • if convergence > largestConvergence, then set largestConvergence = convergence
        • } /* end for each possible state */
    • if largestConvergence $< 10^{-6}$ then we have converged
    }
```

Several comments are in order regarding this algorithm. First, the superscript $^{@}$ denotes locations where boundary conditions must be checked. If the system cannot accept more arrivals of that corresponding class than the price must be set to the maximum to ensure an arrival rate equal to zero. Next, the set of prices ($u_{opt}(1)$ and $u_{opt}(2)$) that prevails when the system converges is the optimal policy. The final value of $h(n_1,n_2)$ indicates the total amount of money we would lose (not receive as revenue) if we started the process in state (n_1,n_2) instead of state $(n_1=0,n_2=0)$. The final value of h_{best0} is the optimal average revenue per stage. This final variable, h_{best0}, is the value that we can compare to the average revenue from an optimal static pricing policy to determine which policy performed better. A comparison of these prices will be shown in the analysis after we develop the procedure to find optimal static prices. Remember the results from [PT99], however – optimal static pricing is nearly as good as optimal dynamic pricing in these situations (when the demand function is known with certainty).

4.3 Demand State Estimation

The heuristic of Section 4.1 relies upon the ability to compute the most likely demand state for each service class. In the state estimation procedure described here we use the information available (interarrival/departure times per class) to calculate the posterior probability that the arrival (or lack of arrivals), together with the price, occurred as a result of a specific demand function. Specifically, Bayes' rule is used to update the probability distribution for the demand state of service class *i* whenever a class *i* request arrives or whenever a class *i* customer departs. We now derive the update equations.

Derivation of the Update Rule Following a Request Arrival:

Let t indicate the time of the most recent arrival in class i,
τ indicate the interarrival time of the most recent arrival in class i,
t - τ be the time of the next most recent arrival,
ds1 indicate the demand state for class 1,
ds2 indicate the demand state for class 2,
H indicate the past history of past arrival times and prices used up to including the t-τ arrival
P(ds=i) = the probability that the demand state of the arriving class is in state i at time t-τ,
P(ds′=i) = the probability that the demand state of the arriving class is in state i at time t,
u_i = the price of service for class i at time t-τ.

We wish to compute P(ds′=i), i=(1,2,3). Note that

$$P(ds'=i\,|\,\mathrm{H},\tau)=\frac{P(ds'=i,\tau\,|\,\mathrm{H})}{P(\tau\,|\,\mathrm{H})}$$

$$=\frac{\sum_{k=1}^{3}P(ds'=i,\tau\,|\,\mathrm{H},ds=k)P(ds=k\,|\,\mathrm{H})}{P(\tau\,|\,\mathrm{H})}$$

$$=\frac{\sum_{k=1}^{3}P(ds'=i,\tau\,|\,ds=k,u)P(ds=k\,|\,\mathrm{H})}{P(\tau\,|\,\mathrm{H})}$$

$$\frac{\sum_{k=1}^{3}P(ds'=i\,|\,ds=k,\tau,u)P(\tau\,|\,ds=k,u)P(ds=k\,|\,\mathrm{H})}{P(\tau\,|\,\mathrm{H})}$$

and let $g_{k,i}(\tau)=P(ds'=i\,|\,ds=k,\tau,u)$

$f_{k,u}(\tau)=P(\tau\,|\,ds=k,u)$

where

$f_{k,u}(\tau)=-\lambda\, e^{-\tau\lambda}$

Probability of demand state transitioning to state k during time τ (which is actually independent of u)

Probability density for the interarrival time τ given demand state k and the price u

To find a value of $g_{ki}(\tau)$ we will make the assumption that this value can be approximated by the probability of transition from state k to i multiplied by the time period, τ. This assumption will be valid as long as τ is much greater than the inverse of the transition rate. In this way we will only be considering the probability of one transition directly from state k to state i. These approximations are as follows:

$$g_{0,0}(\tau)\cong 1-\alpha_{[class],0}*\tau$$
$$g_{0,1}(\tau)\cong \alpha_{[class],0}*\tau$$
$$g_{0,2}(\tau)\cong 0$$
$$g_{1,0}(\tau)\cong \beta_{[class],1}*\tau$$
$$g_{1,1}(\tau)\cong 1-\alpha_{[class],1}*\tau-\beta_{[class],1}*\tau$$
$$g_{1,2}(\tau)\cong \alpha_{[class],1}*\tau$$
$$g_{2,0}(\tau)\cong 0$$
$$g_{2,1}(\tau)\cong \beta_{[class],2}*\tau$$
$$g_{2,2}(\tau)\cong 1-\beta_{[class],2}*\tau$$

Finally, we choose *i* as the most likely demand state if

$$P(c = i \mid x) > P(c = j \mid x)$$
for all $i \neq j$ and Break Ties Arbitrarily (BTA)

Derivation of the Update Rule Following a Departure:

In order to calculate this probability we use the Poisson mass function instead of the exponential density function. Only the changes to the above formulation are shown:

- τ indicates the time period since the last arrival if a class i user,
- P(ds=i) = the probability that the demand state of the departing class is in state i at time t-τ.
- P(ds′=i) = the probability that the demand state of the departing class is in state i at time t.

$$f_{k,u}(\tau) = \frac{e^{-\tau\lambda}\lambda^{o}}{0!} = e^{-\tau\lambda}$$

We choose *i* as the most likely demand state if

$$P(c = i \mid x) > P(c = j \mid x)$$
for all $i \neq j$ and Break Ties Arbitrarily (BTA)

Factoring Revenue into State Estimation:

In order to further increase our revenue we may include cost ratios in our calculations. This will influence the system's perceptions towards the demand states which produce a higher average revenue. To do this the only modification required is to choose *i* as the most likely demand state if

$$P(c = i \mid x) J_{ds1,\, ds2,\, i}{}^* > P(c = j \mid x)\, J_{ds1,\, ds2,\, j}{}^* \quad \text{for all}\, i \neq j \ \text{(BTA)}$$

where Jds1,ds2,I* is the optimal average revenue given the currently perceived state in the non-arriving/departing class and ds=I in the arriving/departing class.

Chapter 5. On Computing Optimal Static Prices

The preceding chapter described a heuristic solution for the POMDP model for the case of partially observed, stochastically varying demand. We measure the value of this heuristic by comparing the revenue it achieves to the revenue associated with optimal static prices. If the heuristic is significantly better, then we have established a nontrivial gap between optimal dynamic and optimal static pricing for the case of uncertain, stochastically varying demand.

In Section 5.1 below, we outline our methodology for computing optimal static prices for the case of known, time-invariant demand. The same numerical technique could have been used by Paschalidis and Tsitsiklis in [PT99] in their computational examples. In Section 5.2, we outline the extensions to this methodology required to compute optimal static prices for the case of partially observed, stochastically varying demand.

5.1 Optimal Static Prices for the Case of Known, Time-Invariant Demand

Optimal static Prices for the MDPs can be computed in much the same way as the optimal dynamic policies. We simply need to change the order of the 'for' loops to check each price pair, and then keep track of the optimal policy. The adapted algorithm is as follows:

```
for every possible price for class 1 ($u_1$)
{
        for every possible price for class 2 ($u_2$)
        {
                while convergence has not occurred
                {
```

```
                for each possible state $(n_1, n_2)$ @
                {
                        solve
                        $htemp = \sum_{i \notin C(N)} \lambda_i(u_i)u_i + \sum_{i \notin C(N)} \frac{\lambda_i(u_i)}{\upsilon} h(N+e_i)$
                                $+ \sum_{i=1}^{M} \frac{n_i \mu_i}{\upsilon} h(N-e_i) + (1 - \sum_{i \notin C(N)} \frac{\lambda_i(u_i)}{\upsilon} - \sum_{i=1}^{M} \frac{n_i \mu_i}{\upsilon}) h(N)$
                        save as $h_{best}$ the highest value in this iteration
                        save as $u_{opt}(1)$ and $u_{opt}(2)$ the prices which achieved that
                optimum
                        if $n_1 = 0$ and $n_2 = 0$, then set $h_{best0} = h_{temp}$
                } /* end for each possible state */
                set $h(n1,n2) = h_{best} - h_{best0}$
                set convergence = Absolute Value ( $h(n1,n2)_{last\ iteration} - h(n1,n2)$ )
                if largest convergence $< 10^{-6}$ then we have converged for this price set
                if $h_{best} > h_{total\ best}$ then $h_{total\ best} = h_{best}$ and uopt1= $u_{opt}(1)$ and uopt2=
                        $u_{opt}(2)$
        } /* end for $u_2$ */
    } /* end for $u_1$ */
}
```

@ indicates a location where boundary conditions must be checked to ensure negative arrival times do not occur

The results of this algorithm are similar to before, with $h_{total\ best}$ being the optimal revenue and uopt1 and uopt2 being the optimal static price policy.

5.2 POMDP Reformulation

Optimal static prices for the case of partially observed, stochastically varying demand are somewhat more complicated to compute, but the computations are at least tractable. Even though the demand states for the system are unobserved, we are able to evaluate the effect of static prices simply by evaluating the average reward for the associated finite state Markov chain. (Prices do not change in response to state transitions, so neither does demand.) To construct the algorithm we must first re-examine several of the equations we used earlier Sections 2.5 and 5.1, specifically (1) the calculation of ν, the maximum transition rate, and (2) the form of Bellman's equation.

Maximum Transition Rate:

In order to calculate the maximum transition rate we must include terms which contain the fastest possible transition rate between demand states.

$$\nu = \sum_{i=1}^{M}\left[\lambda_{o,i} + \mu_i\left(\frac{R}{r_i}\right) + \overline{\alpha_i} + \overline{\beta_i}\right]$$

$$\text{where } \overline{\alpha_i} = \max_{j=0}^{2}(\alpha_{i,j})$$

$$\text{and } \overline{\beta_i} = \max_{j=0}^{2}(\beta_{i,j})$$

Bellman's equation:

Likewise, the Bellman equation must be modified to include the transitions between demand states:

$$\begin{aligned}
&J^* + h(ds_1, ds_2, n_1, n_2) = \\
&\max_{u\in U}\Big[\ \lambda_1(ds_1,u_1)u_1 + \lambda_1(ds_2,u_2)u_2 \\
&+ \frac{\lambda_i(ds_i,u_i)}{\upsilon} h(ds_1,ds_2,n_1+1,n_2) + \frac{\lambda_i(ds_i,u_i)}{\upsilon} h(ds_1,ds_2,n_{1d},n_2+1) \\
&+ \frac{n_i\mu_i}{\upsilon} h(ds_1,ds_2,n_1-1,n_2) + \frac{n_i\mu_i}{\upsilon} h(ds_1,ds_2,n_1,n_2-1) \\
&+ \frac{\alpha_{1,1}}{\upsilon} h(ds_1+1,ds_2,n_1,n_2) + \frac{\alpha_{2,1}}{\upsilon} h(ds_1,ds_2+1,n_1,n_2) \\
&+ \frac{\beta_{1,1}}{\upsilon} h(ds_1-1,ds_2,n_1,n_2) + \frac{\beta_{2,1}}{\upsilon} h(ds_1,ds_2-1,n_1,n_2) \\
&+ \Big(1 - \frac{\lambda_1(ds_1,u_1)}{\upsilon} - \frac{\lambda_2(ds_2,u_2)}{\upsilon} - \frac{n_i\mu_i}{\upsilon} - \frac{n_i\mu_i}{\upsilon} \\
&\qquad - \frac{\alpha_{1,1}}{\upsilon} - \frac{\alpha_{2,1}}{\upsilon} - \frac{\beta_{1,1}}{\upsilon} - \frac{\beta_{2,1}}{\upsilon}\Big) * h(ds_1,ds_2,n_1,n_2)\Big]
\end{aligned}$$

From this we can rewrite our earlier algorithm and now solve the POMDP for optimal static prices:

```
for every possible price for class 1 (u1)
{
for every possible price for class 2 (u2)
```

```
{
    while convergence has not occurred
    {
        for each possible demand state for class one ($ds_1$)
        {
        for each possible demand state for class two ($ds_2$)
        {
            for each possible state ($n_1$, $n_2$) @
            {
        solve
            $htemp = \max_{u \in U} \Big[ \lambda_1(ds_1,u_1)u_1 + \lambda_1(ds_2,u_2)u_2$
            $+ \frac{\lambda_i(ds_i,u_i)}{\upsilon} h(ds_1,ds_2,n_1+1,n_2) + \frac{\lambda_i(ds_i,u_i)}{\upsilon} h(ds_1,ds_2,n_{1d},n_2+1)$
            $+ \frac{n_i\mu_i}{\upsilon} h(ds_1,ds_2,n_1-1,n_2) + \frac{n_i\mu_i}{\upsilon} h(ds_1,ds_2,n_1,n_2-1)$
            $+ \frac{\alpha_{1,1}}{\upsilon} h(ds_1+1,ds_2,n_1,n_2) + \frac{\alpha_{2,1}}{\upsilon} h(ds_1,ds_2+1,n_1,n_2)$
            $+ \frac{\beta_{1,1}}{\upsilon} h(ds_1-1,ds_2,n_1,n_2) + \frac{\beta_{2,1}}{\upsilon} h(ds_1,ds_2-1,n_1,n_2)$
            $+ \Big(1 - \frac{\lambda_1(ds_1,u_1)}{\upsilon} - \frac{\lambda_2(ds_2,u_2)}{\upsilon} - \frac{n_i\mu_i}{\upsilon} - \frac{n_i\mu_i}{\upsilon}$
                $- \frac{\alpha_{1,1}}{\upsilon} - \frac{\alpha_{2,1}}{\upsilon} - \frac{\beta_{1,1}}{\upsilon} - \frac{\beta_{2,1}}{\upsilon}\Big) * h(ds_1,ds_2,n_1,n_2) \Big]$
    and save as $h_{best}$ the highest value in this iteration
                    and save as $u_{opt}(1)$ and $u_{opt}(2)$ the prices
                    which achieved that optimum
                and if $n_1 = n_2 = ds_1 = ds_2 = 0$ set
                $h_{best0} = h_{temp}$
            } /* end for each possible state ($n_1$, $n_2$) */
        } /* end for each demand state ($ds_2$) */
        } /* end for each demand state ($ds_2$) */
            set $h(n1,n2) = h_{best} - h_{best0}$
            set convergence = Absolute Value ( $h(n1,n2)_{\text{last iteration}} - h(n1,n2)$ )
            if largest convergence $< 10^{-6}$ then we have converged
            if $h_{best} > h_{total\ best}$ then $h_{total\ best} = h_{best}$ and uopt1= $u_{opt}(1)$ and uopt2=
                $u_{opt}(2)$
        } /* end for $u_2$ */
    } /* end for $u_1$ */
```

@ indicates a location where boundary conditions must be checked to ensure negative arrival times do not occur

The results of this algorithm are similar to before, with $h_{total\ best}$ being the optimal revenue and uopt1 and uopt2 being the optimal static price policy.

Chapter 6. Examples, Part 1

In this chapter we present the results from three computational examples. We focus on results for the case of perfectly observed, time-invariant demand. Our goal is to illustrate the observation that there is very little gain from optimal dynamic pricing compared to optimal static pricing, at least when the demand functions are known and time-invariant.

6.1 Example I

To test our code and to verify our understanding of the problem, we have computed optimal dynamic and stationary policies for some of the test data described in [PT99].

6.1.1 Problem Data

This application has two classes, each of which has parameters as discussed earlier. Class one demand models are denoted as the number 1 followed by A, B, or C, and class two demand models by the number 2 followed by E, F, or G. A complete model denotes the demand of both classes, such that '1A2E ' would indicate class one is in demand state A while class two is in demand state E.

Parameters for the first example are drawn from an expansion of the parameters from [PT99]. These parameters are based on the working of an ATM network, and rates are in terms of arrivals/departures per hour:

Model	R	a1	b1	a2	b2
1A2E	155	40	4	350	35
1A2F	155	40	4	425	42.5
1A2G	155	40	4	500	50
1B2E	155	60	6	350	35
1B2F	155	60	6	425	42.5
1B2G	155	60	6	500	50
1C2E	155	80	5	350	35
1C2F	155	80	5	425	42.5
1C2G	155	80	8	500	50
	10	40	4	350	35
	155	40	4	350	35
	155	70	4	550	35

For all cases R=155 Mbps, r_1=4 Mbps, r_2=1 Mbps, and μ_1=1 / hr , and μ_2=2 / hr.

Table 6-1 Example I: Model Parameters

6.1.2 Results from the DP Calculations

The following results were obtained from our calculations. This first set compares calculations made by [PT99] to calculations made in our research.

						P & T calculations						Dynamic vs P&T	
Model	R	a1	b1	a2	b2	J* (P&T)	Js(P&T)	% improv	J*	Js	% improv	Difference	% Diff
1A2E	155	40	4	350	35	952.63	945.79	0.72%	950.10	942.86	0.77%	2.53	0.27%
1A2F	155	40	4	425	42.5				1116.66	1105.50	1.01%		
1A2G	155	40	4	500	50	1281.65	1270.40	0.89%	1275.54	1258.71	1.34%	6.11	0.48%
1B2E	155	60	6	350	35				964.41	954.17	1.07%		
1B2F	155	60	6	425	42.5				1124.50	1108.03	1.49%		
1B2G	155	60	6	500	50				1279.02	1251.66	2.19%		
1C2E	155	80	5	350	35	977.28	965.33	1.24%	972.93	954.35	1.95%	4.35	0.45%
1C2F	155	80	5	425	42.5				1129.36	1112.95	1.47%		
1C2G	155	80	8	500	50	1288.97	1273.90	1.18%	1281.24	1258.86	1.78%	7.73	0.60%
	10	40	4	350	35	164.63	163.73	0.55%	164.45	163.45	0.61%	0.18	0.11%
	155	40	4	350	35	952.63	945.79	0.72%	950.10	942.86	0.77%	2.53	0.27%
	155	70	4	550	35	2189.20	2164.40	1.15%	2184.57	2159.44	1.16%	4.63	0.21%

For all cases R=155 Mbps, r_1=4 Mbps, r_2=1 Mbps, and μ_1=1 / hr , and μ_2=2 / hr. For those models where there are no values for [PT99] it is because those models are solely ours and are shown here because of similarity to these models.

Table 6-2 Example I: MDP Optimal Dynamic and Static Pricing

Analysis of these results shows two items of interest:

1. Our calculations differ slightly. The difference can be explained by the price increment used and possibly by selection of convergence criteria. We incremented the prices at 1 dollar while [PT99] increment at 1 cent. The smaller increment they used would result in higher revenue in both the dynamic and static cases. The convergence criteria we employed was to continue execution until convergence was less than 10^{-6} or for 20,000

iterations of the dynamic programming algorithm, whichever came sooner. All of our results in these cases resulted from convergence of 10^{-6}, while the convergence criteria used by [PT99] is unknown, though assumed to be similar. A larger convergence criteria would result in higher values of revenue.

2. We can confirm the analysis that static prices are nearly optimal in the case of many small users and a known demand function.

6.1.3 Verification of Results Through Simulation

To confirm these prices we ran the static and dynamic policies shown here through a simulation (which will be discussed more fully in the next section). The comparison to simulated revenues using these static and dynamic policies for selected models is as follows:

	Dynamic Policy Comparison			Static Policy Comparison		
Model	Solution J*	Simulation Mean	Standard Deviation	Solution Js	Simulation Mean	Standard Deviation
1A2E	950.10	950.04	1.1836	942.86	942.87	1.8523
1B2F	1124.50	1124.69	1.3597	1108.03	1107.36	1.4995
1C2G	1281.24	1281.30	1.3024	1258.86	1258.51	1.6631

Each simulation was run for thirty replications.

Table 6-3 Example I: Simulation Validation

6.2 Example II

The second example is a single channel from a broadband hybrid fiber-coax network running the DOCSIS protocol. The bandwidth available is 27 Mbps, and rates are expressed as the number of transitions per second.

6.2.1. Problem Data

This application again has two classes, each of which has parameters as discussed earlier. Parameters for this sample application are as follows:

Model	a1	b1	a2	b2
1A2E	20	2	10	3
1A2F	20	2	8	3
1A2G	20	2	6	3
1B2E	15	2	10	3
1B2F	15	2	8	3
1B2G	15	2	6	3
1C2E	10	2	10	3
1C2F	10	2	8	3
1C2G	10	2	6	3

For all cases R=27 Mbps , r_1=2 Mbps , r_2=.5 Mbps, μ_1=.000139 / second , and μ_2=.000278 /second.

Table 6-4 Example II: Model Parameters

6.2.2 Results from the DP Calculations

The data which was shown in the heuristic development, section 4.1, was also solved for both optimal dynamic and static policies. The results are as follows:

Model	a1	b1	a2	b2	J*	Js	% improv
1A2E	20	2	10	3	0.05	0.05	0.00%
1A2F	20	2	8	3	0.04	0.04	0.03%
1A2G	20	2	6	3	0.03	0.03	0.73%
1B2E	15	2	10	3	0.05	0.05	0.00%
1B2F	15	2	8	3	0.04	0.04	0.35%
1B2G	15	2	6	3	0.03	0.03	0.57%
1C2E	10	2	10	3	0.05	0.05	0.00%
1C2F	10	2	8	3	0.04	0.04	0.35%
1C2G	10	2	6	3	0.03	0.03	0.87%

For all cases R=27 Mbps , r_1=2 Mbps , r_2=.5 Mbps,
μ_1=.000139 / second , and μ_2=.000278 /second.

Table 6-5 Example II: MDP Optimal Dynamic and Static Pricing

In this case the optimal static performed as well as or nearly as well as the optimal dynamic in all cases. Note that in theses models the service time was very long, as is shown by the very small value of μ. As a result of this and the effective bandwidth requirements compared to the total available, the number of users in the system at one time was small – in fact the largest number of group one users possible was 13 and the largest number of group two users possible was 54. The system tended to be monopolized by group two users at all times for all models.

6.2.3 Verification of Results Through Simulation

The simulated validation of selected models was as follows:

	Dynamic Policy Comparison			Static Policy Comparison		
Model	Solution J*	Simulation Mean	Standard Deviation	Solution Js	Simulation Mean	Standard Deviation
1A2E	0.0487	0.0487	0.0001	0.0487	0.0487	0.0001
1B2F	0.0376	0.0375	0.0000	0.0375	0.0375	0.0000
1C2G	0.0265	0.0263	0.0003	0.0263	0.0263	0.0001

Table 6-6 Example II: Simulation Validation

6.3 Example III

The third and final example we list is very similar to Example II – however in this case the service times were switched between user class one and two. Instead of having the class which required more effective bandwidth (four times as much) taking longer for service, they would now take less time.

6.3.1 Problem Data

This data is the same as that for example two:

Model	a1	b1	a2	b2
1A2E	20	2	10	3
1A2F	20	2	8	3
1A2G	20	2	6	3
1B2E	15	2	10	3
1B2F	15	2	8	3
1B2G	15	2	6	3
1C2E	10	2	10	3
1C2F	10	2	8	3
1C2G	10	2	6	3

For all cases R=27 Mbps , r_1=2 Mbps , r_2=.5 Mbps, μ_1=.000278 / second , and μ_2=.000139 /second.

Table 6-7 Example III: Model Parameters

6.3.2 Results from the DP Calculations

For all cases R=27 Mbps , r_1=2 Mbps , r_2=.5 Mbps,

R	a1	b1	a2	b2	J*	Js	% improv
1A2E	20	2	10	3	0.02	0.02	0.00%
1A2F	20	2	8	3	0.02	0.02	0.00%
1A2G	20	2	6	3	0.02	0.02	2.53%
1B2E	15	2	10	3	0.02	0.02	0.00%
1B2F	15	2	8	3	0.02	0.02	0.00%
1B2G	15	2	6	3	0.01	0.01	3.40%
1C2E	10	2	10	3	0.02	0.02	0.00%
1C2F	10	2	8	3	0.02	0.02	0.00%
1C2G	10	2	6	3	0.01	0.01	0.00%

μ_1=.000278 / second , and μ_2=.000139 /second.

Table 6-8 Example III: MDP Optimal Dynamic and Static Pricing

The results are very similar to the last model – little to no improvement when utilizing dynamic pricing. This is, however, the expectation from such models. We will now go on and develop the simulation for our heuristic and show the gains possible when we do not know with complete accuracy the current demand model.

Chapter 7. Examples, Part 2

In this chapter we will show the computational results and calculations of the heuristic and how it compares to an optimal static pricing policy.

7.1 Comparison Methodology

We developed a simulation which was used to estimate the average revenue we would obtain when utilizing this heuristic, which can be compared to the optimal static prices which were generated for the same models. Specific details of the simulation appear in Appendix A-4.

Random numbers were generated by a procedure which combined two linear congruential generators and also included a shuffle of the values to ensure that serial correlations are broken up. This generator was originally developed by L'Ecuyer and is taken from [PTVF94] entitled 'ran2'. The period of this generator is approximately 2.3 * 10^{18} while the greatest number of random numbers used by any of the simulation runs is approximately 1.5 * 10^{9}. The random numbers used all came from the same seed, and in each run of the complete simulation the same values were assigned to the same variables – thus in each simulation that was executed the arrival time, departure times, and demand state transitions occurred at the same times (in other words, we used common random numbers). This random variable generator was initialized only once at the beginning of each simulation (it was not reset between replications).

The average run length for each replication in the simulation was two years. This value was chosen because it is of a significant length of time to allow multiple transitions through each of the demand states. In addition the first replication was discarded to ensure initial bias did not influence the system (initial bias exists because we start the system in an empty state (n1=n2=0) and the initial average revenue rate is much higher as the system fills to capacity quickly. The following plot shows this relationship:

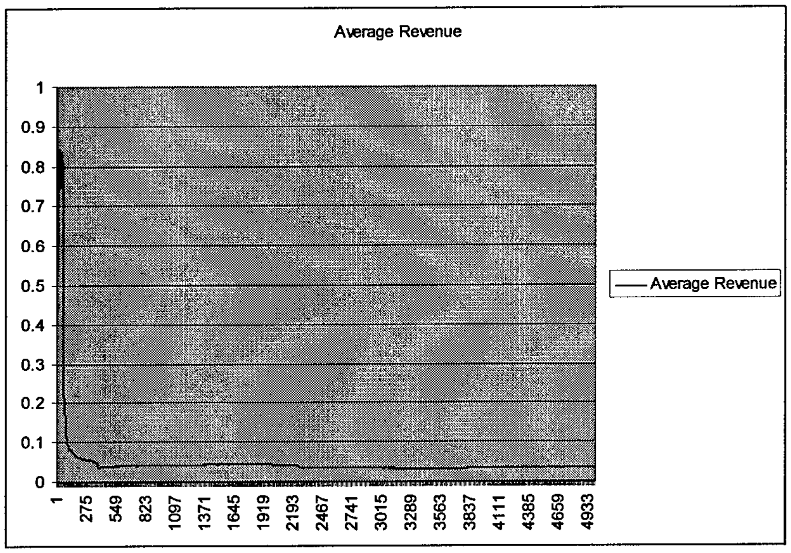

Figure 7.1 Initialization Bias

When the simulation began the x axis is at zero, and as it progresses the average value converges to a steady state value. At the right end of the X axis on the chart the system is in steady state and we can then begin to gather statistics which are not biased. The method from this point on was to run the simulation for an additional two years, capture the statistics of interest, and then reset them to gather statistics for the next replication.

31 total replications were ran of each model, of which the last 30 were used to generate the sample mean and variance for the case being modeled. A hypothesis test was then done as follows:

let $\overline{X}$ = simulation sample mean
S = sample variance
X' = Optimal static pricing policy average revenue

H$_0$: $\overline{X} = X'$

H$_1$: $\overline{X} > X'$

The test statistic we will use is as follows:

$$t_0 = \frac{\overline{X} - X'}{S / \sqrt{n}}$$

where our criteria for rejection is

$$t_0 > t_{\alpha, n-1}$$

We will use an α value of .05, enabling us to obtain a 95% confidence interval.

If we fail to reject H$_0$ we will not be able to ascertain that we have made an improvement, however if we reject H$_0$ we can be 95% certain that we have increased the average revenue by utilization of our heuristic.

7.2 Example I

Below are shown the heuristic output and the optimal static price output for Example I, which, as was mentioned previously, is very similar to the models used by [PT99].

Optimal Dynamic versus Optimal Static Pricing – Summary Statistics:

Dynamic Policy		Optimal Static Policy		
Mean	Standard Deviation	Average Revenue	Actual Increase	Percent Increase
1119.1022	2.0233	975.3089	143.7933	14.74%

Table 7-1 Example I: comparison of Dynamic and Static Pricing

Hypothesis testing:

$t_o = 389.259$

$t_{\alpha,n-1} = 1.699$

Since $t_o > t_{\alpha,n-1},$ we reject the null hypothesis.

Rate of Correct State Estimation: 62 %

Steady State Probability Comparison:

	From Simulation	Actual
P(ds1=0)	0.3331	0.333333
P(ds1=1)	0.3339	0.333333
P(ds1=2)	0.333	0.333333
P(ds2=0)	0.3322	0.333333
P(ds2=1)	0.3337	0.333333
P(ds2=2)	0.3341	0.333333

Table 7-2 Example I: Steady State Probabilities

Alternative Demand State Estimation Technique: The gain in revenue which resulted from the addition of the cost considerations in the classification portion of the heuristic are as follows:

Including Costs		Not Including Costs			
Mean	Standard Deviation	Mean	Standard Deviation	Actual Increase	Percent Increase
1119.1022	2.0233	1117.0523	6.5426	2.0499	0.18%

Table 7-3 Example I: Alternative Demand State Estimation Results

In the following section we will conduct a hypothesis test to determine the significance of this increase.

Verification of Optimal Static Pricing: The optimal static pricing policy results (which was computed from the algorithm developed in Chapter 5) were also validated by the simulation with the following results.

Static Policy Validation		
Solution	Simulated Result	
Js	Mean	Std Dev
975.3089	970.7465	4.0615

Table 7-4 Example I: Static Policy Validation

7.3 Example II

Example II represents the approximate bandwidth available from a single channel of a cable TV company which is utilizing the DOCSIS protocol.

Optimal Dynamic versus Optimal Static Pricing – Summary Statistics:

Dynamic Policy		Optimal Static Policy		
Mean	Standard Deviation	Average Revenue	Actual Increase	Percent Increase
0.0362	0.0008	0.0204	0.0158	77.64%

Table 7-5 Example II: comparison of Dynamic and Static Pricing

Hypothesis testing:

$t_o = 114.8547$

$t_{\alpha,n-1} = 1.699$

Since $t_o > t_{\alpha,n-1}$, we reject the null hypothesis.

Rate of Correct State Estimation: 96.45

Steady State Probability Comparison:

	From Simulation	Actual
P(ds1=0)	0.335209	0.333333
P(ds1=1)	0.331458	0.333333
P(ds1=2)	0.333333	0.333333
P(ds2=0)	0.326625	0.333333
P(ds2=1)	0.338911	0.333333
P(ds2=2)	0.334464	0.333333

Table 7-6 Example II: Steady State Probabilities

Alternative Demand State Estimation Technique: The gain in revenue which resulted from the addition of the cost considerations in the classification portion of the heuristic is as follows.

Including Costs		Not Including Costs			
Mean	Standard Deviation	Mean	Standard Deviation	Actual Increase	Percent Increase
0.036176	0.0008	0.036085	0.0008	0.0001	0.25%

Table 7-7 Example II: Alternative Demand State Estimation Results

In the following section we will conduct an hypothesis test to determine the significance of this increase.

Verification of Optimal Static Pricing: The optimal static pricing policy results (which was computed from the algorithm developed in Chapter 5) were also validated by the simulation with the following results:

Static Policy Validation		
Solution	Simulated Result	
Js	Mean	Std Dev
0.0204	0.020399	0.00105

Table 7-8 Example I: Static Policy Validation

7.4 Example III

Example III is identical to Example II except for the service time distribution.

Optimal Dynamic versus Optimal Static Pricing – Summary Statistics:

Dynamic Policy		Optimal Static Policy		
Mean	Standard Deviation	Average Revenue	Actual Increase	Percent Increase
0.0217	0.0003	0.0196	0.0020	10.24%

Table 7-9 Example III: comparison of Dynamic and Static Pricing

Hypothesis testing:

$t_o = 33.31014$

$t_{\alpha,n-1} = 1.699$

Since $t_o > t_{\alpha,n-1}$, we reject the null hypothesis.

Rate of Correct State Estimation: 95.90%

Steady State Probability Comparison:

	From Simulation	Actual
P(ds1=0)	0.324210	0.333333
P(ds1=1)	0.336201	0.333333
P(ds1=2)	0.339581	0.333333
P(ds2=0)	0.327826	0.333333
P(ds2=1)	0.333080	0.333333
P(ds2=2)	0.339094	0.333333

Table 7-10 Example III: Steady State Probabilities

7.5 Discussion:

The results from Sections 7.2-7.4 are summarized in the table below.

	Dynamic Policy		Optimal Static Policy				
Ex.	Mean	Standard Deviation	Average Revenue	Actual Increase	Percent Increase	to	t,.05,30
I	1119.1022	2.0223	975.3089	143.7933	14.74%	385.28	1.699
II	0.0362	0.0008	0.0204	0.0158	77.64%	114.85	1.699
III	0.0217	0.0003	0.0196	0.0020	10.24%	33.31	1.699

Table 7-11 Consolidated Results – Dynamic vs. Static Pricing

In all three Examples we have failed to reject the null hypothesis, and therefore we can conclude with 95% certainty that the heuristic produces a revenue increase. Indeed in Example II the increase in revenue is substantial. This would amount to over a half a million dollars a year. Indeed, assuming a time interval for Example 1 of one hour (as opposed to seconds for examples 2 and 3) this gain in revenue is also over half a million dollars a year. Example 3 would provide a gain of only around $96,000.

The gain experienced in Example 2 is by far the greatest. This model also has the highest average demand given the relationship between the demand rates and service times. Example 3 is very close to Example 2 in demand functions, but the difference in the relationship of the service times to the effective bandwidth appears to have had significant impact. This relationship can be captured to some extent by a parameter [PT99] used to distinguish demand strength, ρ:

$$\rho = \frac{\lambda_{0,i} * r_i}{R * \mu_i}$$

This parameter captures a measure of the average demand. By looking at the values of ρ for the models in Case 2 the lower values of ρ are in the class which is dominant in the revenue growth (class 2) , while in case 3 the lower values of ρ are in the class which is less dominant.

Values of ρ:

	Case 2		Case 3	
Model	Class 1	Class 2	Class 1	Class 2
1A2E	10667	667	1333	5333
1A2F	10667	533	1333	4267
1A2G	10667	400	1333	3200
1B2E	8000	667	1000	5333
1B2F	8000	533	1000	4267
1B2G	8000	400	1000	3200
1C2E	5333	667	667	5333
1C2F	5333	533	667	4267
1C2G	5333	400	667	3200

Table 7-12 Calculation of ρ

The values of ρ for Case 1 were approximately one for all models, a significant difference from cases 2 and 3.

The utilization of the cost ratios in the classification portion of the heuristic appears to have provided a modest gain in performance. To conclude whether or not this is true we conducted another hypothesis test, which in this case is slightly different than the previous test. This time our test will be as follows:

let $\overline{X_i}$ = simulation sample mean

S_i = sample variance

where

i = 1 if using the cost ratios in classification,

i =2 otherwise (not using cost ratio)

H_0: $\overline{X_1} = \overline{X_2}$

H_1: $\overline{X_1} > \overline{X_2}$

Our test statistic:

$$t_0 = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

where our criteria for rejection is

$t_0 > t_{\alpha,\nu}$

$$\text{where } \nu = \frac{\left(\frac{S_1^2}{n_2} + \frac{S_2^2}{n_2}\right)^2}{\frac{\left(\frac{S_1^2}{n_1}\right)^2}{(n_1+1)} + \frac{\left(\frac{S_2^2}{n_2}\right)^2}{(n_2+1)}} - 2$$

The reason we are using this test now is that we are comparing the mean of two processes of which the variances are unknown and unequal. The reason that the variances are unequal is that by changing the classification rule we change the underlying process.

The result:

	Including Costs		Not Including Costs			
Example	Mean	Standard Deviation	Mean	Standard Deviation	Actual Increase	Percent Increase
1	1119.10	2.02	1117.05	6.54	2.0499	0.18%
2	0.036176	0.0008	0.036085	0.0008	0.0001	0.25%

Table 7-13 Consolidated Results – Comparison of Alternate Estimation Procedure

Example	to	nu	t,alpha,nu
1	1.639494	36.87567	1.688
2	0.452	62	1.67

Table 7-14 Hypothesis Tests of Comparison

In this instance we fail to reject the hypothesis for both cases, and must conclude that the addition of the cost ratios in the heuristic does not provide a significant increase in revenue.

Chapter 8. Conclusions and Recommendations

8.1 Conclusions

This work explores the relationship between optimal static and sub-optimal dynamic pricing in an environment where demand is unknown and changing in time. To accomplish this we have developed a heuristic that examines the allocation of a network resource, bandwidth, at the user level (albeit somewhat grossly, with the assumption of effective bandwidth greatly simplifying our work). This heuristic solves a finite number of perfect information Markov decision problems, one for each demand state profile, and then uses these solutions along with partial observations of the demand state gained from user arrivals to generate a dynamic pricing policy for the network that is capable of gains in revenue of up to 80% over the revenue of an optimal statically priced system.

This work contributes to the large amount of recent research into network pricing which primarily examines the concept from a static point of view. Most analysis in this area of research assumes a constant load upon the system, and does explicitly accommodate the stochastic nature of arrivals and departures, as does our heuristic. In the more common formulation of the system demand, the instantaneous rate of request arrivals is assumed to be a constant and known function of price, and prices are adjusted to find an optimal mix of the user classes. Our formulation takes into account the stochastic nature of demand and takes advantage of the gains in revenue which can be made from fluctuations in this demand. We feel that this framework used to model demand is likely to be more realistic in that it can account for unpredictable periods of higher and lower demand.

This heuristic is similar to Open Loop Feedback Control [B95] which generates a suboptimal solution to a POMDP by examining the state of the system upon each partial observation and enacting the control which would be optimal on the basis that no further observations will become available to update the system.

Additionally, we have developed a state estimation algorithm to determine which of the demand states is most likely for each service class. This estimation algorithm is capable of correct classification rates as high as 96%, with the lowest observed rate being 62%. In all cases the prior probabilities of the class distribution were equal to 1/3, thus the classification algorithm provides significant benefit in determining the state of the system.

8.2 Recommendations for Future Work

We believe the next step in this research will be to expand the study to additional models to attempt to determine a relationship between the gain possible by the heuristic and the input parameters of the demand functions, service times, and bandwidth requirements. It may be possible to generalize the results in terms of these parameters to establish a quick prediction of the gains possible from implementation of the heuristic. These insights could be of great value in other application domains where resource allocation can be done from an economic perspective.

We believe the next essential step following this will involve the modeling of actual network demand to determine the extent to which this heuristic can be used in increasing revenue. The work by [KA98] into on-line estimation of linear demand functions for network services could be extended to this model to gain some certainty about the nature of the demand functions. From this we could determine the amount of gain to expect from these actual parameters.

Following this, the work should be expanded by incorporating explicit models for statistical multiplexing in networks, where fluctuations in bandwidth requirements are considered. In this way a more accurate analysis of network qualities of service can be conducted. Adaptation of this heuristic to include the constraints imposed by service guarantees would provide a complete heuristic which could be implemented on any network.

This type of formulation can be used to optimize alternative metrics, beyond the revenue achieved for anetwork service provider. [PT99] used a very similar formulation to analyze a measure of network welfare. Another possible application is in military networks where it would be appropriate to maximize a measure of the priority of information that traverses a network. The goal here would be to minimize the congestion due to the excessive transmission of relatively unimportant information.

BIBLIOGRAPHY

[B95] D. P. Bertsekas. Dynamic Programming and Optimal Control. Vol. I. & II. Athena Scientific, 1995.

[B99] Y. Bernet, et al. Differentiated Services. Working Paper, Internet Engineering Task Force (IETF), 1999

[BRS99] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. Proceedings ACM SIGCOMM, 1999.

[CKL94] A. R. Cassandra, L. P. Kaelbing, and M. L. Littman. Acting Optimally in Partially Observable Stochastic Domains. Working Paper CS-94-20, Department of Computer Science, Brown University. April 1994

[CP99] A. Croll and E. Packman. Managing Bandwidth: Deploying QoS in Enterprise Networks, Prentice Hall, 1999.

[E86] H. K. Eldeib. Multistage Decision Making with Imprecise Information: Mathematical Modeling and Computational Procedures. Dissertation. Department of Systems Engineering, University of Virginia, 1986.

[GC98] H. R. Gibson and B. Com. Easing the Internet Traffic Jam: A Comparison of Traffic Expansion and Congestion Tolls as Competing alternatives for Reducing Internet Congestion. Thesis. The University of Newcastle, Australia. 1998

[GSW94] A. Gupta, D. O. Stahl, and A. B. Whinston. An Economic approach to Network Computing with Priority classes. The University of Texas at Austin, 1995.

[GSW95] A. Gupta, D. O. Stahl, and A. B. Whinston. The Internet: A future tragedy Of The Commons? The University of Texas at Austin, 1995.

[GSW95a] A. Gupta, D. O. Stahl, and A. B. Whinston. Pricing of Services on the Internet. The University of Texas at Austin, 1995.

[GSW95b] A. Gupta, D. O. Stahl, and A. B. Whinston. A Stochastic Equilibrium Model of Internet Pricing. The University of Texas at Austin, 1995.

[GSW95d] A. Gupta, D. O. Stahl, and A. B. Whinston. Priority Pricing of Integrated Services Networks. The University of Texas at Austin, 1995.

[H74] A. Hordijk. Dynamic Programming and Markov Potential Theory. Mathematisch Cetrum, 1974.

[HM80] W. Hines and D. Montgomery. Probability and Statistics in Engineering and Management Science. John Wiley and Sons, 1980.

[KA98] N. Keon and G Anandalingam. Real Time Pricing of Multiple Telecommunications Services Under Uncertain demand. Working Paper 98-13, Department of Systems Engineering, University of Pennsylvania, 1998.

[KA99] N. Keon and G Anandalingam. Adaptive flow Control for VoD Using Price Discounts. Department of Systems Engineering, University of Pennsylvania, 1998.

[KG99] F. Kelly and R. Gibbens. Pricing the Internet: Resource Pricing and the Evolution of Congestion Control. http://www.statslab.cam.ac.uk/~djw1005/Stats/Compete/

[KM99] P. B. Key and D. R. McAuley. differential QoS and Pricing in Networks: where flow-control meets game theory. IEE Proceedings Software, March 1999.

[KMB99] P. B. Key, D. R. McAuley, and P. Barham. Congestion Pricing for Congestion Avoidance. Technical Report, Microsoft Research, Cambridge, UK 1999.

[L90] J. Lark. Applications of Best-First Heuristic Search to Finite-Horizon Partially Observed Markov Decision Processes. Dissertation. Department of Systems Engineering, University of Virginia, 1990.

[LK91] A. M. Law and W. D. Kelton. Simulation Modeling and Analysis. McGraw-Hill, 1991.

[LS83] L. Ljung and T. Soderstrom. Theory and Practice of Recursive Identification. MIT Press, 1983.

[L90] W. Lovejoy An Approximate Algorithm, With Bounds, For Composite State Partially Observed Markov Decision Problems. Proceedings, 29th Conference on Decision and Control, IEEE 1990.

[LSP87] R. Lipsey, P. Steiner, and D. Purvis. Economics. Harper & Row, 1987.

[MB99] C.R. McConnnell and S.L. Brue. Economics: Principles, Problems, and Policies. 14th Edition. Irwin McGraw-Hill, 1999.

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm. Computer Communications Review, Vol 27, No. 3, July 1997.

[O97] A. Odlyzko. A Modest Proposal for Preventing Internet Congestion. DIMACS Technical Report 97-68. AT&T Labs-Research, 1997.

[PTVF94]W. Press, s. Teukolsky, W. Vettering, B. Flannery. Numerical Recipes in C, Second Edition. Cambridge University Press, 1994.

[PT99] I Ch. Paschaldis and J. N. Tsitsiklis. Congestion-Dependent Pricing of Network Services. Technical Report, Systems Group, Department of Manufacturing Engineering, Boston University, 1998.

[R99] Ripley, B.D. Pattern Recognition and Neural Networks. University Press, Cambridge, 1999.

[RS83] Ross, S.M. Stochastic Processes. John Wiley & Sons, Inc. 1983.

[S78] E. Sondik. The Optimal control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs. Operations Research, 1978.

[T96] A. S. Tanenbaum. Computer Networks. Prentice Hall, 1996.

A Computer Code

A-1: Simulation of Heuristic: Pseudo code

The following is the pseudo code for the simulation:

```
/*  simulation of heuristic */
/*  establish new random number generator */
/*  define variables to use in program*/
/*  assign values to variables */
/*  establish all possible system states
        if (n[1]*r[1]) + (n[2]*r[2]) <= R state is possible
/*  Read in optimal dynamic pricing policies from MDP solutions   */
/*  generate starting system data */
/*  set initial prior probabilities */
/*  generate random objects and set seed */
Repeat the following for 31 replications:
    while not converged:
            /* generate arrival rates for classes  based on current prices and hidden Markov chain
            /* randomnly generate arrival times for both classes
            /* generate departure rates for classes  based on meu[class] * n[class]
            /* randomnly generate departure times for both classes
            /* randomnly generate transition times for classes for increase and decrease of demand
            /* select the event with the earliest occurrence and cancel all other transaction
    if a departure is going to occur:
                        /*      update time;
                        /*      update system state;
                        /*  execute classification rule if system is not at full load:
                            (if ((n[1]*r[1] + n[2]*r[2] ) < (Bandwidth - r[departclass]) )
                        /*      set new price based upon updated system state;
                        /*  goto while not converged
    if a shift demand state is going to occur:
                        /*      update time;
                        /*      update hidden Markov Chain
                        /*  goto while not converged
    if an arrival is going to occur:
                        /*      update time;/
                        /*      update revenue based upon current price
                        /*      calculate average revenue (total revenue / time)
                        /*      update system state;
                        /*      execute classification rule
                        /*      set new price based upon updated system state;
                        /*  goto while not converged
     } /* end while not converged
            Write statistics on replication to output files
} /* end for 31 replications
            Calculate overall statistics and write to output file
END SIMULATION.
```

A-2: Simulation of Heuristic: Pseudo code

Following is the actual C code for the simulation:

```
/* simulation of heuristic */
/* FILE CURRENTLY SET UP To
run Example 2. */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
/*    establish new random number generator */
      #define IM1 2147483563
      #define IM2 2147483399
      #define AM (1.0/IM1)
      #define IMM1 (IM1-1)
      #define IA1 40014
      #define IA2 40692
      #define IQ1 53668
      #define IQ2 52774
      #define IR1 12211
      #define IR2 3791
      #define NTAB 32
      #define NDIV (1+IMM1/NTAB)
      #define EPS 1.2e-7
      #define RNMX (1.0-EPS)
      float random(long *idum)
      {             int e;
      long k;
      static long idum2=123456789;
      static long iy=0;
      static long iv[NTAB];
      float temp;
      if (*idum <= 0) {
                    if (-(*idum)<1)*idum=1;
                    else *idum=-(*idum);
                    idum2 = (*idum);
                    for (e=NTAB+7; e>=0;e--){
                                  k=(*idum)/IQ1;
                                  *idum=IA1*(*idum-k*IQ1)-k*IR1;
                                  if (*idum < 0) *idum  += IM1;
                                  if (e<NTAB) iv[e] = *idum;
                    }
                    iy=iv[0];
                    }
                    k=(*idum)/IQ1;
                    *idum=IA1*(*idum-k*IQ1)-k*IR1;
                    if (*idum <0) *idum += IM1;
                    k = idum2/IQ2;
                    idum2=IA2*(idum2-k*IQ2)-k*IR2;
                    if (idum2<0) idum2 += IM2;
                    e=iy/NDIV;
                    iy=iv[e]-idum2;
                    iv[e]=*idum;
                    if (iy < 1) iy += IMM1;
                    if ((temp=AM*iy) > RNMX) return RNMX;
                    else return temp;
             }
main()
{
/* define variables to use in program*/
      int M = 2;                           /* Number of service classes */
      double Bandwidth;             /* Total bandwidth avail (in Mbps) */
      double r[3];                         /* required by service class */
      double a[3][3];                      /* demand equation parameters [class][demand state]*/
      double b[3][3];                      /* demand equation parameters [class][demand state]*/
      int nmax1 = 13;                      /* max number in system class 1 */
int nmax2 = 54;                     /* value = FLOOR(R / r[i]) */
int    N[15][56];              /* system states*/
double h[4][3][15][56];        /* revenue stream expected in demand state ds1 & ds2 from n1 n2 state */
int nm1[3][3];
int nm2[3][3];
double u1[3][3][15][56];       /* optimal prices obtained from data files for [ds][n1][n2] */
double u2[3][3][15][56];
int aa;
int bb;
int l;
int ll;
double gh;
double converg;
double avgrtn[3][3];
double newnum[3];
double newdenom;
double newprob[3][3];
double ucurrent[3];
double var;
double increment;              /* amount that price is incremented by */
double meu[3];                      /* service rate for class i */
double alfa[3][3];             /* transition rate (increasing) for state i from demand state ds */
double beta[3][3];             /* transition rate (decreasing) for state i from demand state ds */
double newd1;
int ds[3];                          /* demand state */
int pds[3];                         /* perceived demand state */
int t;
int s;
double prob[3][3];
double g[3][3];
double f[3];
double lambdalam[3];
double conv;
double tau;                         /* interarrival time */
double totalrevenue;
double      averagerevenue;    /* totalrevenue / time */
double oldaverage;
double time;                        /* sim clock */
int n[3];
double iteration;
double q;
long arrive[3];                     /* random number hlder for arrival of [class] */
double      arriveuniform[3];  /* normalized random number between 0 and 1 */
double arrivaltime[3];
double lambda[3];              /* current arrival rate [class] */
int arriveclass;
int shiftup[3];                     /* random number hlder for shift in demand state for [class]*/
double shiftuptime[3];
double      shiftupuniform[3];  /* normalized random number between 0 and 1 */
int shiftupclass;
int shiftdown[3];                   /* random number hlder for shift in demand state for [class]*/
double shiftdowntime[3];
double      shiftdownuniform[3];  /* normalized random number between 0 and 1 */
int shiftdownclass;
int depart[3];
double departtime[3];
double departuniform[3];
double departlambda[3];
int departclass;
int shift2;
int shift1;
int i;
int j;
char c;
long fl;
float fop;
char x[81];
char y;
double classright;
double classtotal;
```

```
        double classrate;
        int replication;
        double oldtime;
        double oldtime1;
        double timeds11;
        double probds11;
        double timeds12;
        double probds12;
        double timeds10;
        double probds10;
        double timeds21;
        double probds21;
        double timeds22;
        double probds22;
        double timeds20;
        double probds20;
        double avgrevcalc[31];
        double sigmatotal;
        double nef;
        double classdptright;
        double classdpttotal;
        double totaltime;
        long lon;
                FILE *fp;
                FILE *stream;
                FILE *out;
                FILE *classification;
                FILE *classi;
                FILE *rep0;
                FILE *checkdmd;


        classi = fopen("departclass.txt", "w+");
        checkdmd = fopen("chkdmd1.txt", "w+");
        out = fopen("SHORTRUN.txt", "w+");
        classification = fopen("classification with costs.txt", "w+");
        rep0 = fopen("replication0 stats.txt", "w+");
        fprintf( checkdmd, "time at ds1 for class 1:\t");
        fprintf( out, "replication \t" );
        fprintf( out, "time \t" );
        fprintf( out, "iteration\t"  );
        fprintf( out, "converg\t"  );
        fprintf( out, " averagerevenue\t" );
        fprintf( out, "classright\t" );
        fprintf( out, "classtotal\t"  );
        fprintf( out, " classrate\n" );
        fprintf( classification, "arrclas: " );
        fprintf( classification, " tau \t"  );
        fprintf( classification, "ucurrent \t"  );
        fprintf( classification, "prob[0]: \t" );
        fprintf( classification, "prob[1]: \t");
        fprintf( classification, "prob[2]: \t" );
        fprintf( classification, "pds[i]: \t" );
        fprintf( classification, "ds[i]: \t" );
        fprintf( classification, "correct? \n" );
        fprintf( classi, "dptclas: " );
        fprintf( classi, " tau \t"  );
        fprintf( classi, "ucurrent \t"  );
        fprintf( classi, "prob[0]: \t" );
        fprintf( classi, "prob[1]: \t");
        fprintf( classi, "prob[2]: \t" );
        fprintf( classi, "pds[i]: \t" );
        fprintf( classi, "ds[i]: \t" );
        fprintf( classi, "correct? \n" );
for (i = 0; i<=30; i++)
avgrevcalc[i] = 0;
sigmatotal = 0;
  fp    = fopen( "1A2E.txt", "r+" );
  stream = fopen( "1A2E.txt", "r+" );
/*  assign values to variables */
```

```
        a[1][0] = 20;/* a value class i demand state ds */
        a[2][0] = 10;
        a[1][1] = 15;
        a[2][1] = 8;
        a[1][2] = 10;
        a[2][2] = 6;
        b[1][0] = 2.0;
        b[2][0] = 3.0;
        b[1][1] = 2.0;
        b[2][1] = 3.0;
        b[1][2] = 2.0;
        b[2][2] = 3.0;
                r[1] = 2;                       /* bandwidth required by service class */
                r[2] = .5;
                meu[1]=.000139;                 /* service rate for a sertvice time = 2 hrs */
                meu[2]=.000278;                 /* service rate for a sertvice time = 1 hr */
                alfa[1][0] = .0000695; /* transition rate for an increase in 4 hours */
                alfa[1][1] = .0000695; /* transition rate for an increase in 4 hours */
                alfa[1][2] = 0; /* not a possible transition */
                alfa[2][0] = .0000926;  /* transition rate for an increase in 3 hours */
                alfa[2][1] = .0000926;  /* transition rate for an increase in 3 hours */
                alfa[2][2] = 0; /* not a possible transition */
                beta[1][0] = 0; /* not a possible transition */
                beta[1][1] = .0000695; /* transition rate for an decrease in 6 hours */
                beta[1][2] = .0000695; /* transition rate for an decrease in 6 hours */
                beta[2][0] = 0; /* not a possible transition */
                beta[2][1] = .0000926;  /* transition rate for an decrease in 3 hours */
                beta[2][2] = .0000926;  /* transition rate for an decrease in 3 hours */
                increment = .25;        /* prices incremented by */
                Bandwidth = 27;                 /* R FOR A CABLE NETWORK - USING 1 DOWNSTREAM
CHANNEL */
/* establish all possible system states
 set N[][] = 1 if state possible o.w. = 0 */
                for (i = 0; i <= nmax1; i++)
                {
                        n[1] = i;
                        for (j = 0; j <= nmax2; j++)
                        {
                                n[2] = j;
                                var = (n[1]*r[1]) + (n[2]*r[2]);
                                if ( var  <= Bandwidth)
                                        N[n[1]][n[2]] = 1;
                                        else N[n[1]][n[2]] = 0;
                        }       /* end for j */
                }       /* end for i */
        N[nmax1+1][nmax2] = 0;
        N[nmax1][nmax2+1] = 0;
        N[nmax1+1][nmax2+1] = 0;
/*              Read in optimal pricing data from 9 solved problems:   */
 for (t=0; t<=2; t++)
{
        for (s=0; s<=2; s++)
        {
                if ((t==0) && (s==0))
                        fp = fopen("1A2E.txt","r+");
                else if ((t==0)&&(s==1))
                        fp = fopen("1A2F.txt","r+");
                else if ((t==0)&&(s==2))
                        fp = fopen("1A2G.txt","r+");
                else if ((t==1) && (s==0))
                        fp = fopen("1B2E.txt","r+");
                else if ((t==1)&&(s==1))
                        fp = fopen("1B2F.txt","r+");
                else if ((t==1)&&(s==2))
                        fp = fopen("1B2G.txt","r+");
                else if ((t==2) && (s==0))
                        fp = fopen("1C2E.txt","r+");
                else if ((t==2)&&(s==1))
                        fp = fopen("1C2F.txt","r+");
                else if ((t==2)&&(s==2))
```

```
                        fp = fopen("1C2G.txt","r+");
                if( fp == NULL )
                {
                        printf( "The file  was not opened\n" );
                        printf("t= %3d s = %3d \n", t,s);
                        printf("\n \n");
                        c=getchar();
                }
                else
                {
                        iteration = 0;
                        fseek( fp, 0L, SEEK_SET );
                        fscanf( fp, "%lf", &iteration );
                        fscanf( fp, " %d", &nm1[t][s] );
                        fscanf( fp, " %d", &nm2[t][s] );
                        fscanf( fp, " %lf", &avgrtn[t][s] );
                        for (aa=0; aa <= nm1[t][s]; aa++)
                        {
                                for (bb=0; bb <= nm2[t][s]; bb++)
                                {
                                        fscanf( fp, "%d", &l );
                                        fscanf( fp, "%d", &ll );
                                        fscanf( fp, "%d", &N[l][ll] );
                                        fscanf( fp, "%lf", &h[t][s][l][ll] );
                                        fscanf( fp, "%lf", &u1[t][s][l][ll] );
                                        fscanf( fp, "%lf", &u2[t][s][l][ll] );
                                        fscanf( stream, "%s", x );
                                }
                        }
        fclose(fp);
                }
        }       /*  end for s     */
}       /*  end for t     */
/*              generate starting system data */
                time = 0;
                n[1] = 0;               /* reusing variable as the current number in [class] */
                n[2] = 0;
                ds[1] = 0;
                ds[2] = 0;
                pds[1] = 0;
                pds[2] = 0;
                time = 0;
                totalrevenue = 0;
                iteration = 0;
                ucurrent[1]=1;
                ucurrent[2]=1;
                classright=0;
                classtotal=0;
                classdptright=0;
                classdpttotal=0;
                totaltime = 0;
                timeds11 = 0;
                timeds12 = 0;
                timeds10 = 0;
                timeds21 = 0;
                timeds22 = 0;
                timeds20 = 0;
        /* set initial prior probabilities */
                prob[1][0] = (1.0/3.0);
                prob[1][1] = 1.0/3.0;
                prob[1][2] = 1.0/3.0;
                prob[2][0] = (1.0/3.0);
                prob[2][1] = 1.0/3.0;
                prob[2][2] = 1.0/3.0;
/*      generate random objects and set seed */
                lon = 7;
/* generate  shiftup rates for classes and departure rates b*/
for (replication = 0; replication <=31; replication++)
{
converg = 99999;
```

```
oldtime = 0;
oldtime1=0;
                conv = 1;
                while (conv > 0)
                {
                          iteration++;
                          tau=99999999;
/* generate arrival rates for classes based on current prices */
                lambda[1] = a[1][ds[1]] - b[1][ds[1]]*ucurrent[1];
                lambda[2] = a[2][ds[2]] - (b[2][ds[2]]*ucurrent[2]);
if ((lambda[1] < 0) || (lambda[2] < 0))
{
     printf("a[i][ds[i]]: %6.6f\t %6.6f\n ",a[1][ds[1]],a[2][ds[2]]);
     printf("b[i][ds[i]]: %6.6f\t %6.6f\n ",b[1][ds[1]],b[2][ds[2]]);
     printf("ds[i]: %3d\t %3d\n ",ds[1],ds[2]);
     printf("ucurrent[i]: %6.12f\t %6.12f\n ",ucurrent[1],ucurrent[2]);
     printf("lambda[i]: %6.12f\t %6.12f\n \n",lambda[1],lambda[2]);
}
/* generate departure rates for classes based on meu[class] * n[class] */
                departlambda[1] = meu[1] * n[1];
                departlambda[2] = meu[2] * n[2];          /*
/*              generate arrival times or passage times for each attribute (num in sys, class 1 & 2, ds1&2 */
if (N[n[1]+1][n[2]] == 1)
{               arriveuniform[1] = random(&lon);
                arrivaltime[1] = - (1/lambda[1]) * log(arriveuniform[1]);
}
else arrivaltime[1] = 99999999;
if (N[n[1]][n[2]+1] == 1)
{
                arriveuniform[2] = random(&lon);
                arrivaltime[2] = - (1/lambda[2]) * log(arriveuniform[2]);
}
else arrivaltime[2] = 99999999;
                if (ds[1] <= 1)
                {
                          shiftupuniform[1] = random(&lon);
                          shiftuptime[1] = - (1/alfa[1][ds[1]]) * log(shiftupuniform[1]);
                }
                else shiftuptime[1] = 999999999;
                if (ds[2] <= 1)
                {
                          shiftupuniform[2] = random(&lon);
                          shiftuptime[2] = - (1/alfa[2][ds[2]]) * log(shiftupuniform[2]);
                }
                else shiftuptime[2] = 999999999;
                if (ds[1] >= 1)
                {
                          shiftdownuniform[1] = random(&lon);
                          shiftdowntime[1] = - (1/beta[1][ds[1]]) * log(shiftdownuniform[1]);
                }
                else shiftdowntime[1] = 999999999;
                if (ds[2] >= 1)
                {
                          shiftdownuniform[2] = random(&lon);
                          shiftdowntime[2] = - (1/beta[2][ds[2]]) * log(shiftdownuniform[2]);
                }
                else shiftdowntime[2] = 999999999;
if (n[1] > 0)
{
                departuniform[1] = random(&lon);
                departtime[1] = - (1/departlambda[1]) * log(departuniform[1]);
}
else departtime[1] = 99999999;
if (n[2] > 0)
{               departuniform[2] = random(&lon);
                departtime[2] = - (1/departlambda[2]) * log(departuniform[2]);
}
else departtime[2] = 99999999;
/*              calculate which time is first - set equal to tau */
                          shiftupclass = 0;
```

```
                        departclass = 0;
                        arriveclass = 0;
                        shiftdownclass = 0;
                if (arrivaltime[1]<arrivaltime[2])
                {       tau = arrivaltime[1];
                        arriveclass = 1;
                }
                        else
                        {       tau = arrivaltime[2];
                        arriveclass = 2;        }
                if (departtime[1]<tau)
                {       tau = departtime[1];
                        arriveclass = 0;
                        departclass = 1;        }
                if (departtime[2]<tau)
                {       tau = departtime[2];
                        arriveclass = 0;
                        departclass = 2;        }
                if (shiftuptime[1]<tau)
                {       departclass = 0;
                        arriveclass = 0;
                        tau = shiftuptime[1];
                        shiftupclass = 1;               }
                if (shiftuptime[2]<tau)
                {       tau = shiftuptime[2];
                        departclass = 0;
                        arriveclass = 0;
                        shiftupclass = 2;               }
                if (shiftdowntime[1]<tau)
                {       shiftupclass = 0;
                        departclass = 0;
                        arriveclass = 0;
                        tau = shiftdowntime[1];
                        shiftdownclass = 1;             }
                if (shiftdowntime[2]<tau)
                {       shiftupclass = 0;
                        departclass = 0;
                        arriveclass = 0;
                        tau = shiftdowntime[2];
                        shiftdownclass = 2;             }
/* if departure:*/
                if (departclass != 0)
                {
                                        /*      update time;*/
                                                time = time + tau;
/* the following will update pds[departureclass] iff the system is not at full load,
  intent is to decrease prob of being in incorrect state and provide a mechanism for
  changing price and pds other than at the arrival of a user */
if ((n[1]*r[1] + n[2]*r[2] ) < (Bandwidth - r[departclass]) )
{
g[1][0] = beta[departclass][1]*tau;
g[1][2] = alfa[departclass][1]*tau;
g[1][1] = 1 - g[1][0] - g[1][2];
g[0][1] = alfa[departclass][0]*tau;
g[0][0] = 1 - g[0][1];
g[0][2] = 0;
g[2][0] = 0;
g[2][1] = beta[departclass][2]*tau;
g[2][2] = 1 - g[2][1];
lambdalam[0] = (a[departclass][0] - b[departclass][0]*ucurrent[departclass]);
lambdalam[1] = (a[departclass][1] - b[departclass][1]*ucurrent[departclass]);
lambdalam[2] = (a[departclass][2] - b[departclass][2]*ucurrent[departclass]);
if (lambdalam[0] > 0)
    f[0] =exp(-tau*lambdalam[0]);
else f[0] = 1;
if (lambdalam[1] > 0)
f[1] = exp(-tau*lambdalam[1]);
else f[1] = 1;
if (lambdalam[2] > 0)
f[2] = exp(-tau*lambdalam[2]);
else f[2] = 1;
 if ((f[0]<0)||(f[1]<0)||(f[1]<0))
        c=getchar();
if ((f[0]>1)||(f[1]>1)||(f[1]>1))
        c=getchar();
newdenom = 0;
                for (i=0; i<=2; i++)
                {
                newd1=g[0][i] * f[0]  *  prob[departclass][0];

                                newdenom = newdenom + (g[0][i] * f[0]  *  prob[departclass][0]  ) + ( g[1][i] * f[1] *
prob[departclass][1]    )
                                                + (g[2][i] * f[2] * prob[departclass][2]    );
                                newnum[i] = (g[0][i] * f[0]  *  prob[departclass][0]  ) + ( g[1][i] * f[1] *
prob[departclass][1]    )
                                                + (g[2][i] * f[2] * prob[departclass][2]    );
                }
if (newdenom != 0)
{
newprob[departclass][0] = newnum[0] / newdenom;
newprob[departclass][1] = newnum[1] / newdenom;
newprob[departclass][2] = newnum[2] / newdenom;
prob[departclass][0] = newprob[departclass][0];
prob[departclass][1] = newprob[departclass][1];
prob[departclass][2] = newprob[departclass][2];
/*add cost factor and classify - see if we can improve:
re-using some variables - must reset after classification rule used */
if (departclass == 1)
{
        prob[departclass][0] = prob[departclass][0]*avgrtn[0][pds[2]];
        prob[departclass][1] = prob[departclass][1]*avgrtn[1][pds[2]];
        prob[departclass][2] = prob[departclass][2]*avgrtn[2][pds[2]];
}
else {
        prob[departclass][0] = prob[departclass][0]*avgrtn[pds[1]][0];
        prob[departclass][1] = prob[departclass][1]*avgrtn[pds[1]][1];
        prob[departclass][2] = prob[departclass][2]*avgrtn[pds[1]][2];
}
/*      classify demand state (highest probability);*/
if ((prob[departclass][0] > prob[departclass][1]) && (prob[departclass][0] > prob[departclass][2]))
        pds[departclass] = 0;
else if ((prob[departclass][1] > prob[departclass][0]) && (prob[departclass][1] > prob[departclass][2]))
        pds[departclass] = 1;
else pds[departclass] = 2;
/*reset priors for next rep:*/
prob[departclass][0] = newprob[departclass][0];
prob[departclass][1] = newprob[departclass][1];
prob[departclass][2] = newprob[departclass][2];
if (replication == 1)
{
        fprintf( classi, "%d\t", departclass );
        fprintf( classi, "%f\t", tau );
        fprintf( classi, "%f\t", ucurrent[departclass] );

        fprintf( classi, "%f\t", prob[departclass][0] );
        fprintf( classi, "%f\t", prob[departclass][1] );
        fprintf( classi, "%f\t", prob[departclass][2] );
        fprintf( classi, "%d\t", pds[departclass] );
        fprintf( classi, "%d\t", ds[departclass] );
        if (pds[departclass] == ds[departclass])
                fprintf( classi, "Y\n" );
        else                    fprintf( classi, "N\n" );
}

if(pds[departclass]             == ds[departclass])
        classdptright=classdptright+1;
classdpttotal=classdpttotal+1;
}       /* if newdenom <> 0 */
}       /* end if nr < B-r */
/*              update state info;*/
```

```
if (departclass == 1)
n[1] = n[1] - 1;
else n[2] = n[2] - 1;
/*     set new price;*/
ucurrent[1] = u1[pds[1]][pds[2]][n[1]][n[2]];
ucurrent[2] = u2[pds[1]][pds[2]][n[1]][n[2]];
if (ucurrent[1] > a[1][ds[1]]/b[1][ds[1]])
              ucurrent[1] = a[1][ds[1]]/b[1][ds[1]];
if (ucurrent[2] > a[2][ds[2]]/b[2][ds[2]])
       ucurrent[2] = a[2][ds[2]]/b[2][ds[2]];
departclass = 0;
} /* end if departclass */
/*                   goto while not converged */

       /* if demand state (shiftup) transition: */
              if (shiftupclass != 0)
              {
       /*            update time;*/
       time = time + tau;
       if (ds[1] == 1)
       {             timeds11 = timeds11 + (time - oldtime1);
       oldtime1 = time;}
       if (ds[1] ==  0)
       {             timeds10 = timeds10 + (time - oldtime1);
       nef = time - oldtime1;
              fprintf( checkdmd, "%f\t", nef);
       oldtime1 = time;}
       if (ds[1] == 2)
       {             timeds12 = timeds12 + (time - oldtime1);
       oldtime1 = time;}
       if (ds[2] == 1)
       {             timeds21 = timeds21 + (time - oldtime);
       oldtime = time;}
       if (ds[2] ==  0)
       {             timeds20 = timeds20 + (time - oldtime);
       oldtime = time;}
       if (ds[2] == 2)
       {             timeds22 = timeds22 + (time - oldtime);
       oldtime = time;}
       shift1 = shift1 + 1;

/*     change arrival rate data (lambda changes based on price in effect)*/
if (shiftupclass == 1)
       ds[1]= ds[1]+1;
else ds[2]=ds[2]+1;
/* if the actual ds changes we must ensure prices are not exceeded */
if (ucurrent[1] > a[1][ds[1]]/b[1][ds[1]])
              ucurrent[1] = a[1][ds[1]]/b[1][ds[1]];

if (ucurrent[2] > a[2][ds[2]]/b[2][ds[2]])
ucurrent[2] = a[2][ds[2]]/b[2][ds[2]];
shiftupclass = 0;
} /*end if shiftupclass */
/*     goto while not converged */

if (shiftdownclass != 0)
{
       /*            update time;*/
       time = time + tau;
       if (ds[1] == 1)
       {             timeds11 = timeds11 + (time - oldtime1);
                     nef = time - oldtime1;
                     fprintf( checkdmd, "%f\t", nef);
       oldtime1 = time;}
       if (ds[1] ==  0)
       {             timeds10 = timeds10 + (time - oldtime1);
       oldtime1 = time;}
       if (ds[1] == 2)
       {             timeds12 = timeds12 + (time - oldtime1);
       oldtime1 = time;}
```

```
        if (ds[2] == 1)
        {               timeds21 = timeds21 + (time - oldtime);
        oldtime = time;}
        if (ds[2] ==  0)
        {               timeds20 = timeds20 + (time - oldtime);
        oldtime = time;}
        if (ds[2] == 2)
        {               timeds22 = timeds22 + (time - oldtime);
        oldtime = time;}
        shift2 = shift2 + 1;

/*      change arrival rate data (lambda changes based on price in effect)*/
if (shiftdownclass == 1)
ds[1]= ds[1]-1;
else ds[2]=ds[2]-1;
/* if the actual ds changes we must ensure prices are not exceeded */
if (ucurrent[1] > a[1][ds[1]]/b[1][ds[1]])
ucurrent[1] = a[1][ds[1]]/b[1][ds[1]];

if (ucurrent[2] > a[2][ds[2]]/b[2][ds[2]])
ucurrent[2] = a[2][ds[2]]/b[2][ds[2]];
shiftdownclass = 0;
} /*end if shiftupclass */


/* if user arrival:*/
if (arriveclass != 0)
{
/*      update time;*/
time = time + tau;
/*      update total revenue and system state*/
totalrevenue = totalrevenue + ucurrent[arriveclass];
n[arriveclass] = n[arriveclass]+1;
/*      calculate average revenue and check for convergence;*/
averagerevenue = totalrevenue / time;
converg = fabs (averagerevenue - oldaverage);
        if  (iteration > 100000)
        {
        printf("converged\n \n ");
        conv = -1;}
        oldaverage = averagerevenue;
/*      calculate prob of being in demand state i;*/
g[1][0] = beta[arriveclass][1]*tau;
g[1][2] = alfa[arriveclass][1]*tau;
g[1][1] = 1 - g[1][0] - g[1][2];
g[0][1] = alfa[arriveclass][0]*tau;
g[0][0] = 1 - g[0][1];
g[0][2] = 0;
g[2][0] = 0;
g[2][1] = beta[arriveclass][2]*tau;
g[2][2] = 1 - g[2][1];
lambdalam[0] = a[arriveclass][0] - b[arriveclass][0]*ucurrent[arriveclass];
lambdalam[1] = a[arriveclass][1] - b[arriveclass][1]*ucurrent[arriveclass];
lambdalam[2] = a[arriveclass][2] - b[arriveclass][2]*ucurrent[arriveclass];
if (lambdalam[0] > 0)
f[0] = lambdalam[0] * exp(-tau*lambdalam[0]);
else f[0] = 0;
if (lambdalam[1] > 0)
f[1] = lambdalam[1] * exp(-tau*lambdalam[1]);
else f[1] = 0;
if (lambdalam[2] > 0)
f[2] = lambdalam[2] * exp(-tau*lambdalam[2]);
else f[2] = 0;

newdenom = 0;
for (i=0; i<=2; i++)
{
newd1=g[0][i] * f[0]  *  prob[arriveclass][0];
newdenom = newdenom + (g[0][i] * f[0]  *  prob[arriveclass][0]   ) + ( g[1][i] * f[1] * prob[arriveclass][1]    )
        + (g[2][i] * f[2] * prob[arriveclass][2]    );
newnum[i] = (g[0][i] * f[0]  *  prob[arriveclass][0]   ) + ( g[1][i] * f[1] * prob[arriveclass][1]    )
        + (g[2][i] * f[2] * prob[arriveclass][2]    );
}
if  (newdenom!=0)
{
newprob[arriveclass][0] = newnum[0] / newdenom;
newprob[arriveclass][1] = newnum[1] / newdenom;
newprob[arriveclass][2] = newnum[2] / newdenom;
prob[arriveclass][0] = newprob[arriveclass][0];
prob[arriveclass][1] = newprob[arriveclass][1];
prob[arriveclass][2] = newprob[arriveclass][2];
/*add cost factor and classify - see if we can improve:
re-using some variables - must reset after classification rule used*/
if (arriveclass == 1)
{
        prob[arriveclass][0] = prob[arriveclass][0]*avgrtn[0][pds[2]];
        prob[arriveclass][1] = prob[arriveclass][1]*avgrtn[1][pds[2]];
        prob[arriveclass][2] = prob[arriveclass][2]*avgrtn[2][pds[2]];
}
else {
        prob[arriveclass][0] = prob[arriveclass][0]*avgrtn[pds[1]][0];
        prob[arriveclass][1] = prob[arriveclass][1]*avgrtn[pds[1]][1];
        prob[arriveclass][2] = prob[arriveclass][2]*avgrtn[pds[1]][2];
}
/*      classify demand state (highest probability);*/
if ((prob[arriveclass][0] > prob[arriveclass][1]) && (prob[arriveclass][0] > prob[arriveclass][2]))
        pds[arriveclass] = 0;
else if ((prob[arriveclass][1] > prob[arriveclass][0]) && (prob[arriveclass][1] > prob[arriveclass][2]))
        pds[arriveclass] = 1;
else pds[arriveclass] = 2;
/*reset priors for next rep:*/
prob[arriveclass][0] = newprob[arriveclass][0];
prob[arriveclass][1] = newprob[arriveclass][1];
prob[arriveclass][2] = newprob[arriveclass][2];
if ((prob[arriveclass][0]<0) || (prob[arriveclass][1]<0) || (prob[arriveclass][2] <0 ))
{
        printf("prob[arriveclass][i]: %f\t %f\t %f\n", prob[arriveclass][0], prob[arriveclass][1], prob[arriveclass][2]);
        printf("neg prob occurred \n \n");
        c=getchar();
}
if (replication == 1)
{
        fprintf( classification, "%d\t", arriveclass );
        fprintf( classification, "%f\t", tau );
        fprintf( classification, "%f\t", ucurrent[arriveclass] );

        fprintf( classification, "%f\t", prob[arriveclass][0] );
        fprintf( classification, "%f\t", prob[arriveclass][1] );
        fprintf( classification, "%f\t", prob[arriveclass][2] );
        fprintf( classification, "%d\t", pds[arriveclass] );
        fprintf( classification, "%d\t", ds[arriveclass] );
        if (pds[arriveclass] == ds[arriveclass])
                fprintf( classification, "Y\n" );
        else                    fprintf( classification, "N\n" );
}
if(pds[arriveclass]             == ds[arriveclass])
        classright=classright+1;
classtotal=classtotal+1;
}
/*      set new price;*/
ucurrent[1] = u1[pds[1]][pds[2]][n[1]][n[2]];
ucurrent[2] = u2[pds[1]][pds[2]][n[1]][n[2]];

if (ucurrent[1] > a[1][ds[1]]/b[1][ds[1]])
ucurrent[1] = a[1][ds[1]]/b[1][ds[1]];

if (ucurrent[2] > a[2][ds[2]]/b[2][ds[2]])
ucurrent[2] = a[2][ds[2]]/b[2][ds[2]];

if (replication == 0)
```

```
{
        fprintf( rep0, "%f\n", averagerevenue);
}
        arriveclass = 0;
} /*end if  arrive class */
/*      goto while not converged */


} /* end while not converged */

totaltime = totaltime + time;
printf(" \n \n replication = %d\n", replication);
printf("%f\t  %f\t secs revenue: %f\t converg: %f\n",iteration, time, averagerevenue, converg);
classrate = classright / classtotal;
printf(" classright = %f\t", classright);
printf(" classtotal = %f\n", classtotal);
printf(" classrate = %6.6f\n", classrate);
        fprintf( out, "%d\t", replication );
        fprintf( out, "%f\t", time );
        fprintf( out, "%f\t", iteration );
        fprintf( out, "%f\t", converg );
        fprintf( out, "%f\t", averagerevenue );
        fprintf( out, "%f\t", classright );
        fprintf( out, "%f\t", classtotal );
        fprintf( out, "%f\t", classrate );
        fprintf( out, "%f\t", classdptright );
        fprintf( out, "%f\t", classdpttotal );
        classrate = classdptright / classdpttotal;
        fprintf( out, "%f\n", classrate );
if (replication != 0)
        avgrevcalc[replication] = avgrevcalc[replication]+ averagerevenue;
        if (ds[1] == 1)
        {               timeds11 = timeds11 + (time - oldtime1);
        if (time < oldtime1)
        {               printf("time: %f\t oldtime1: %f\t rep: %d\n",time, oldtime1, replication);
        c=getchar();}
        oldtime1 = time;}
        if (ds[1] ==  0)
        {               timeds10 = timeds10 + (time - oldtime1);
        oldtime1 = time;}
        if (ds[1] == 2)
        {
        timeds12 = timeds12 + (time - oldtime1);
        oldtime1 = time;}
        if (ds[2] == 1)
        {               timeds21 = timeds21 + (time - oldtime);
        oldtime = time;}
        if (ds[2] ==  0)
        {               timeds20 = timeds20 + (time - oldtime);
        oldtime = time;}
        if (ds[2] == 2)
        {               timeds22 = timeds22 + (time - oldtime);
        oldtime = time;}
/* re-init6ialize before next replication */
time = 0;
totalrevenue = 0;
classright = 0;
classtotal = 0;
classdptright=0;
classdpttotal=0;
iteration = 0;
oldtime = 0;
oldtime1=0;
} /* end for replication */

for (i = 1; i<=30; i++)
avgrevcalc[0] = avgrevcalc[i] + avgrevcalc[0];
avgrevcalc[0]=avgrevcalc[0]/(replication-2);
fprintf( out, "\n mean average value: %f\n", avgrevcalc[0]);
for (i = 1; i<=30; i++)
```

```
sigmatotal = sigmatotal + ( (avgrevcalc[0]-avgrevcalc[i]) * (avgrevcalc[0]-avgrevcalc[i]) );
sigmatotal=sigmatotal/30;                    /* variance */
fprintf( out, "variance = %f\n", sigmatotal);
sigmatotal = sqrt(sigmatotal);               /* std deviation */
fprintf( out, "standard deviation = %f\n", sigmatotal);
probds11= timeds11/totaltime;
probds10= timeds10/totaltime;
probds12= timeds12/totaltime;
probds21= timeds21/totaltime;
probds20= timeds20/totaltime;
probds22= timeds22/totaltime;
oldtime1 = timeds11+timeds10+timeds12;
oldtime = timeds21+timeds20+timeds22;
    fprintf( out, "\n totaltime: %f\n", totaltime );
    fprintf( out, "total time For ds1 %f\t", oldtime1 );
    fprintf( out, "total time For ds2 %f\n \n", oldtime );
    fprintf( out, "timeds10: %f\t", timeds10 );
    fprintf( out, "probds10: %f\n", probds10 );
    fprintf( out, "timeds11: %f\t", timeds11 );
    fprintf( out, "probds11: %f\n", probds11 );
    fprintf( out, "timeds12: %f\t", timeds12 );
    fprintf( out, "probds12: %f\n", probds12 );
    fprintf( out, "timeds20: %f\t", timeds20 );
    fprintf( out, "probds20: %f\n", probds20 );
    fprintf( out, "timeds21: %f\t", timeds21 );
    fprintf( out, "probds21: %f\n", probds21 );
    fprintf( out, "timeds22: %f\t", timeds22 );
    fprintf( out, "probds22: %f\n \n", probds22 );
    fclose(out);
    fclose(classification);
    fclose(rep0);
    fclose(classi);
}           /* end main */
```